НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр КОВАЛЬ
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення розподілених систем»

спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Сервіс live-статистики стану IT-ринку на певний момент чи проміжок часу»

Виконав:
студент IV курсу, групи ТВ-61
Бочок В’ячеслав Олександрович _____

Керівник:
доцент, кандидат технічних наук
Кублій Лариса Іванівна _____

Рецензент:
доцент, кандидат технічних наук
Самарай Валерій Петрович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший, бакалаврський

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр КОВАЛЬ
(підпис)
”___”_____2020р.

ЗАВДАННЯ

на дипломну роботу студенту

_____Бочку В’ячеславу Олександровичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи «Сервіс live-статистики стану IT-ринку на певний момент чи проміжок часу»
керівник роботи Кублій Лариса Іванівна, доцент, кандидат технічних наук
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020 р. № **1168-с**

2. Строк подання студентом роботи 14 червня 2020 р.
3. Вихідні дані до роботи: форма реалізації — веб-сервіс з інтерфейсом для користувача (React), серверною частиною (Python, Flask) і сервісом збору та обробки даних (Scrapy, Python). Средовище розробки — PyCharm.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): проаналізувати існуючий ринок вакансій для інформаційних технологій, проаналізувати існуючі рішення, розробити архітектуру сервісів і їхню взаємодію, реалізувати програмний код, створити конфігурацію для автоматичного розгортання застосунку й забезпечення комунікації між сервісами.
5. Перелік ілюстративного матеріалу: «Завдання розробки сервісу live-статистики стану IT-ринку», «Аналіз існуючих рішень», «Засоби розробки», «Опис програмної

реалізації: Система збору даних», «Опис програмної реалізації: Система обробки даних», «Опис програмної реалізації: Серверна частина», «Опис програмної реалізації: Інтерфейс користувача», «Робота користувача з системою: Налаштування», «Робота користувача з системою: Статистика — фільтри», «Робота користувача з системою: Статистика — інформаційні блоки», «Висновки».

7. Дата видачі завдання «11» жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	11 жовтня 2019 р.	
2.	Вивчення та аналіз задачі	13 квітня 2020 р.	
3.	Розробка архітектури та загальної структури системи	15 квітня 2020 р.	
4.	Розробка структур окремих підсистем	20 квітня 2020 р.	
5.	Програмна реалізація системи	27 квітня 2020 р.	
6.	Оформлення пояснювальної записки	10 травня 2020 р.	
7.	Захист програмного продукту	10 червня 2020 р.	
8.	Передзахист	10 червня 2020 р.	
9.	Захист	15 червня 2020 р.	

Студент _____ _____Бочок В.О._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Кублій Л.І._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою роботи є розробка системи, що надає актуальну статистику щодо стану ринку праці інформаційних технологій, з метою надавати інформацію щодо певних технологій для визначення їх актуальності. В рамках даної дипломної роботи було виявлено існуючі рішення та підходи, оцінено їх сильні та слабкі сторони, а також специфіку вирішення проблем. В ході аналізу було проаналізовано можливі джерела даних для статистики та виявлено найбільш точний у рамках задачі. Згідно зі специфікою джерела було розроблено алгоритми та підходи до виокремлення певної структурованої інформації, на основі якої буде будуватися статистика, а також вміст самої статистики, а саме, сплановано важливі інформативні графіки та таблиці, що матимуть важливе значення для користувача. На основі заданих специфікацій було сплановано архітектуру системи, розбито функціонал на модулі, що взаємодіють один з одним. Для зручного обслуговування системи було описано спеціальний файл з інструкціями, що дозволяють запускати всі сервіси системи в ізольованому середовищі, не турбуючись про інфраструктуру.

Дана дипломна робота обсягом у 78 сторінок, містить 4 додатки та 30 посилань. Також наведено 33 рисунки.

Ключові слова: статистика, вакансії, обробка даних, збір даних, мікросервіси, Docker-compose, Unit-tests.

ABSTRACT

The aim of the work is to develop a system that provides up-to-date statistics on the state of the information technology labor market, in order to provide information on certain technologies to determine their relevance. In the framework of this work, the existing solutions and approaches were identified, their strengths and weaknesses were assessed, as well as the specifics of problem-solving. During the analysis, possible data sources for statistics were analyzed and the most accurate within the problem was identified. According to the specifics of the source, algorithms and approaches to the selection of certain structured information were developed, on the basis of which the statistics will be built, as well as the content of the statistics, namely, important informative graphs and tables that will be important for the user. Based on the specified specifications, the system architecture was planned, the functionality was divided into modules that interact with each other. For easy maintenance of the system, a special file was described with instructions that allow you to run all system services in an isolated environment without worrying about the infrastructure.

This thesis of 78 pages contains 4 appendices and 30 references. There are also 33 figures.

Keywords: statistics, vacancies, data processing, data collection, microservices, Docker-compose, Unit-tests.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IT — інформаційні технології.

Стек — сукупність (в даному контексті сукупність технологій, якими володіє інженер).

Парсер — програмне забезпечення, що здатне виокремлювати інформацію з таких джерел даних, як API, web-сторінки, тощо.

Метрика — оцінка.

Веб-краулінг — процес сканування веб сторінок. Здебільшого агент обробки сам здатний знаходити посилання на нові сторінки і тим самим сканувати весь ресурс.

Скрапінг — процес вилучення даних з різних джерел. У випадку обробки веб сторінок — вимагає також процесу краулінга, що дозволяє знайти всі необхідні сторінки ресурсу.

СКБД — система керування базами даних.

CSS — каскадні таблиці стилів.

ВСТУП

Сучасні інформаційні технології розв’язують дуже широкий спектр задач. Часто певний клас задач накладає свої обмеження й вимагає застосування спеціалізованих інструментів і підходів. Також треба враховувати тенденції щодо збільшення кількості навантаження на систему (кількість запитів на секунду), збільшення обсягів даних. Це змушує знаходити нові підходи. Стрімкий розвиток бізнесу мотивує розробляти не тільки програмні рішення для кінцевих клієнтів, але й інструменти розробки. Це призводить до того, що наразі наявна велика кількість технологій, що є бажаними чи необхідними для розробника, але дуже важко серед них виділити ті, які дають найбільшу перевагу на ринку праці.

Існує кілька підходів до розв’язання цієї проблеми:

1) аналіз наявних вакансій на відкритих платформах пошуку праці;

2) експертна оцінка;

3) ресурси, які надають періодичну (щомісячну, щорічну тощо) статистику.

Перший спосіб обмежений тим, що вимагає ручного періодичного перегляду великої кількості вакансій. Обмеженням є також те, що такі сайти здебільшого не обмежують сферу, для якої пропонують роботу, а отже, не мають спеціальних інструментів, які полегшують пошук. У цьому випадку немає доступу до вакансій, закритих через неактуальність, що не дає змоги оцінити тенденції. Прикладами таких ресурсів є Robota.ua [1], Work.ua [2], Djinni.co [3].

Експертна оцінка сильно залежить від особи, яка її надає. Не завжди зрозуміло, на яких метриках вона базується і чи не є упередженою. До того ж, одна людина не може однаково знатися на всіх сферах і підтримувати свою експертність за умови швидких змін. Також важко робити висновки щодо реального стану ринку праці, оскільки експертних думок може бути велика кількість і вони можуть бути суперечливі.

Гарним прикладом останнього підходу є Dou.ua [4] та Djinni.co . Проблемою таких ресурсів є широка спеціалізованість ресурсів, а також джерела, на основі яких збирається статистика. Для прикладу, статистика Dou збирається згідно з опитуваннями інженерів, що вже призводить до певної неточності і малої вибірки, оскільки опитування проходить порівняно невелика кількість людей, які не мають ніяких стимулів надавати правдиву інформацію. До того ж, через неможливість перевірити деякі відповіді, як наприклад, рівень заробітної платні. Така інформація носить лише рекомендаційний характер. На противагу Dou, Djinni використовує самі вакансії для аналізу, але досить обмежено аналізує зміст. З вакансій виділяється лише одна головна технологія з досить стислого списку (не більше 30), за якою відбувається групування за орієнтовною заробітною платою, а також досвідом роботи. Окремо надається статистика частоти публікацій певної вакансії.

Такі ресурси надають інформацію про конкретні технології, але не про їхнє сумісне використання, що не дає можливості зрозуміти, яких ще знань не вистачає для доповнення вже існуючих. Такі платформи збирають статистику щодо найбільших, базових і добре відомих технологій, ігноруючи нові, які стрімко набувають популярності, або ж розглядають їх окремо чи порівняно з обмеженою кількістю технологій.

Метою проекту є розроблення програмного забезпечення, здатного надавати статистику щодо певних технологій та їхніх об’єднань на основі вакансій для найбільш точного виявлення потреб бізнесу. Таку систему можуть використовувати як інженер, так і викладачі, які розробляють навчальні плани. Така система дасть можливість додати до загальноосвітніх предметів ще й сформовані практичні стеки, а випускники зможуть з найменшими зусиллями і без необхідності додаткового навчання виконувати реальну роботу.

1. ЗАДАЧА ПОБУДОВИ СИСТЕМИ АНАЛІЗУ ВАКАНСІЙ

При виконанні роботи треба:

1) проаналізувати існуючий ринок вакансій для інформаційних технологій:
   а) виявити правила і закономірності;
   б) визначити типову структуру вакансії;
   г) визначити важливість і ступені довіри до інформації з різних джерел;
   ґ) обрати оптимальні джерела інформації;
2) проаналізувати існуючі рішення:
   а) виявити проблеми, які розв’язуються;
   б) проаналізувати переваги й недоліки кожного підходу;
3) розробити архітектуру сервісів і їхню взаємодію;
4) реалізувати програмний код таких сервісів:
   а) парсерів, які збирають дані з Інтернет-ресурсів;
   б) модуля, який обробляє дані;
   в) сервера, який керує взаємодією інших модулів і зв’язком з базами даних;
   г) графічного інтерфейсу для користувача та адміністратора;
5) створити конфігурацію для автоматичного розгортання застосунку й забезпечення комунікації між сервісами.

Пріоритетом при проектуванні й виборі технологій має бути швидкість розробки, простота виконання, а також гнучкість.

Така система є досить складною і містить багато компонентів, тому доцільно більш детально визначити вимоги до кожного сервісу.

1.1 Створення системи збору даних

Основною перевагою над ручним опрацюванням вакансій є можливість машини опрацьовувати більшу їхню кількість за менший час. Така система має

обробляти велику кількість даних без необхідності внесення їх у систему оператором. Згідно з аналізом, найкращим джерелом вакансій є сайти, які надають публічну інформацію про вакансії. Для того, щоб опрацьовувати їх, необхідно побудувати систему з парсерів, які автоматично завантажують веб-сторінки, виділяють потрібну інформацію і зберігають її.

Для реалізації такої системи найкраще підійде фреймворк Scrapy [5] для мови Python. Він вже реалізує багато необхідних механізмів, тож слід розробити лише інтеграцію під ресурси, які надають вакансії [6]. Для цієї роботи було вибрано 2 найбільших в Україні сайти з вакансіями: Djinni та Epam.

Система має завантажувати інформацію про всі доступні вакансії, виділяти її в заздалегідь узгоджену структуру й зберігати у базі даних.

1.2 Створення системи обробки даних

Вакансії надаються у вигляді тексту вільної форми, що значно ускладнює будь-яку обробку і зберігання. Для проведення аналізу треба мати чітку структуру, яку можливо опрацьовувати. Для розв’язання цієї проблеми потрібно розробити систему, яка буде виокремлювати необхідні дані з тексту. Така система має бути досить гнучкою і легко переналаштовуватися, оскільки IT швидко змінюються. Також вона має ігнорувати форматування тексту: абзаци, заголовки, переходи на нові рядки, жирний чи курсивний шрифт, списки й таблиці. Вхідні дані — текст вакансії, вихідні — рівень вакансії та технології, знайдені в тексті.

1.3 Створення серверної частини

Необхідно розробити систему, яка буде контролювати доступ до бази даних. Такою системою має бути сервер REST API, написаний мовою Python з використанням фреймворку Flask [7, 8]. Цей сервіс буде контролювати права доступу, аутентифікацію, доступ до ресурсів і приховувати деталі реалізації зберігання даних [9]. На даний момент сервіс повинен працювати з базами даних

SQLite і MongoDB, але мати можливість з найменшими змінами мігрувати з SQLite на PostgreSQL або MySQL [10, 11].

1.4 Створення інтерфейсу користувача

Інтерфейс системи має інформативно зображувати зібрану статистику, надавати можливість фільтрувати результати, входити в систему, а також підтримувати систему oAuth-аутентифікації [12].

Система має бути гнучкою. Це вимагає побудови досить складних конфігурацій у форматі JSON, що майже неможливо підтримувати людиною. Для розв’язання цієї проблеми має бути розроблений інтерфейс користувача, який надасть можливість створювати й тестувати налаштування, а також завантажувати їх у систему обробки даних. Ця функція має бути доступна лише адміністратору, тим не менш, користувачі мають зберегти можливість переглядати конфігурації системи без можливості щось змінити.

2. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Аналітика потреб ринку роботи в IT — не нова проблема, тому вже існують готові реалізації різних підходів, кожен з яких розв’язує конкретні питання різними способами. Серед найбільш відомих і вдалих є аналітика, яка надається сайтами роботи або незалежними рейтингами. Нижче проаналізовано їхні переваги й недоліки.

2.1 Сайти з пошуку роботи

Перевагами використання сайтів з пошуку роботи є:

1) аналітика націлена на конкурентність і доступність певного класу вакансій;

2) широкий доступ до інформації, яка може бути захована від користувачів (заробітна плата, історія операцій над вакансією, кількість відгуків і відмов, приватні прямі звернення до потенційних робітників), що дає змогу проводити більш точний аналіз;

3) можливість проводити опитування серед робітників і роботодавців.

Недоліки:

1) гнучкість системи не дає можливості впроваджувати інструменти, які були б корисними для розробників. Наприклад, дуже мало ресурсів мають хоча б мінімальний контроль технологій і контроль за якістю певних вакансій;

2) орієнтованість на кількість вакансій певного широкого класу.

Для таких сайтів пріоритетом є впровадження комунікації найбільшої кількості робітників з найбільшою кількістю роботодавців. Надання повної та вичерпної статистики вимагає впровадження додаткових систем, які не відносяться до основної бізнес-логіки і вимагають багато часу й витрат. Саме тому статистика здебільшого є поверхневою. Винятком є сайт Dou.ua, що раз у місяць або півроку надає статистику щодо найпопулярніших мов програмування з прив’язкою до

локації й зарплатні. Джерелом для аналізу є анонімні опитування відвідувачів платформи, що призводить до відносно малої вибірки і часто неточних результатів, оскільки опитані ніяк не зацікавлені в наданні правдивої інформації.

На рисунку 2.1 подано приклад статистики заробітної плати, що надає Djinni . Вона враховує кількість усіх пропозицій від компаній робітнику. Тобто, якщо одному інженеру запропонували роботу 10 різних компаній, то це буде 10 спостережень (без врахування, яку саме пропозицію прийняв робітник). Статистика наводиться для певного міста й певної технології (для прикладу, нижче наведено дані для Python-розробника в м. Києві від 2019-12-03 до 2002-05-31.)

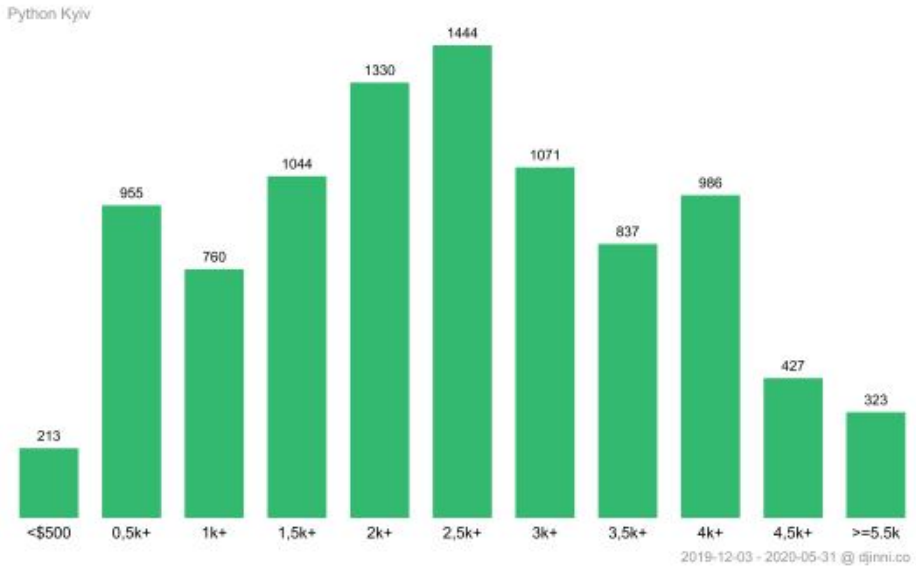


Рисунок 2.1 — Приклад статистики заробітної плати від Djinni [3]

Сервіс Dou надає більше статистики щодо кандидатів і заробітної плати. Наприклад, на рисунку 2.2 наведено динаміку росту заробітної плати залежно від дати. Також сайт Dou надає детальну статистику щодо опитаних (рисунок 2.3).

Розподіл заробітної плати залежно від рівня посади, яку займає інженер, подано на рисунку 2.4.

Сервіс Dou детально описує методологію того, як проводиться відбір спостережень і які дані відображаються користувачеві. Для прикладу наведено розподіл зарплати залежно від вищого навчального закладу випускника (рисунок

2.5). Також зазначено, що всього зібрано 3291 анкета, а також обрано лише ті навчальні заклади, у яких наявні понад 40 анкет.

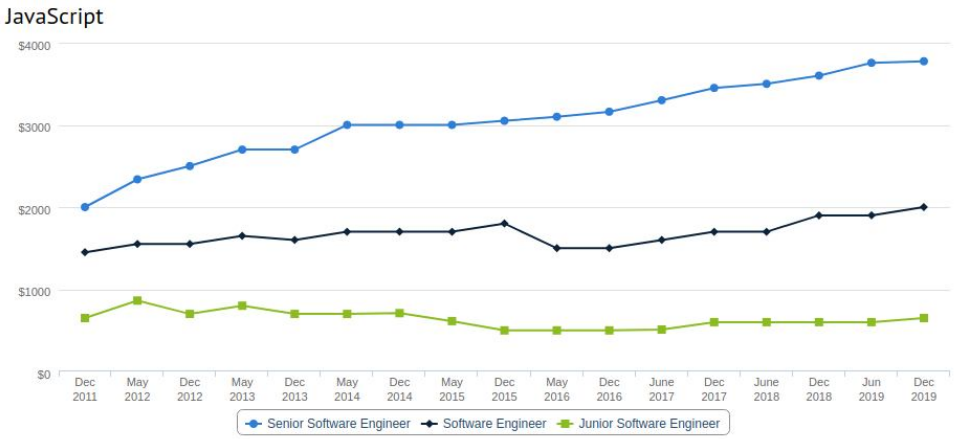


Рисунок 2.2 — Динаміка росту заробітної плати JavaScript-розробників [4]

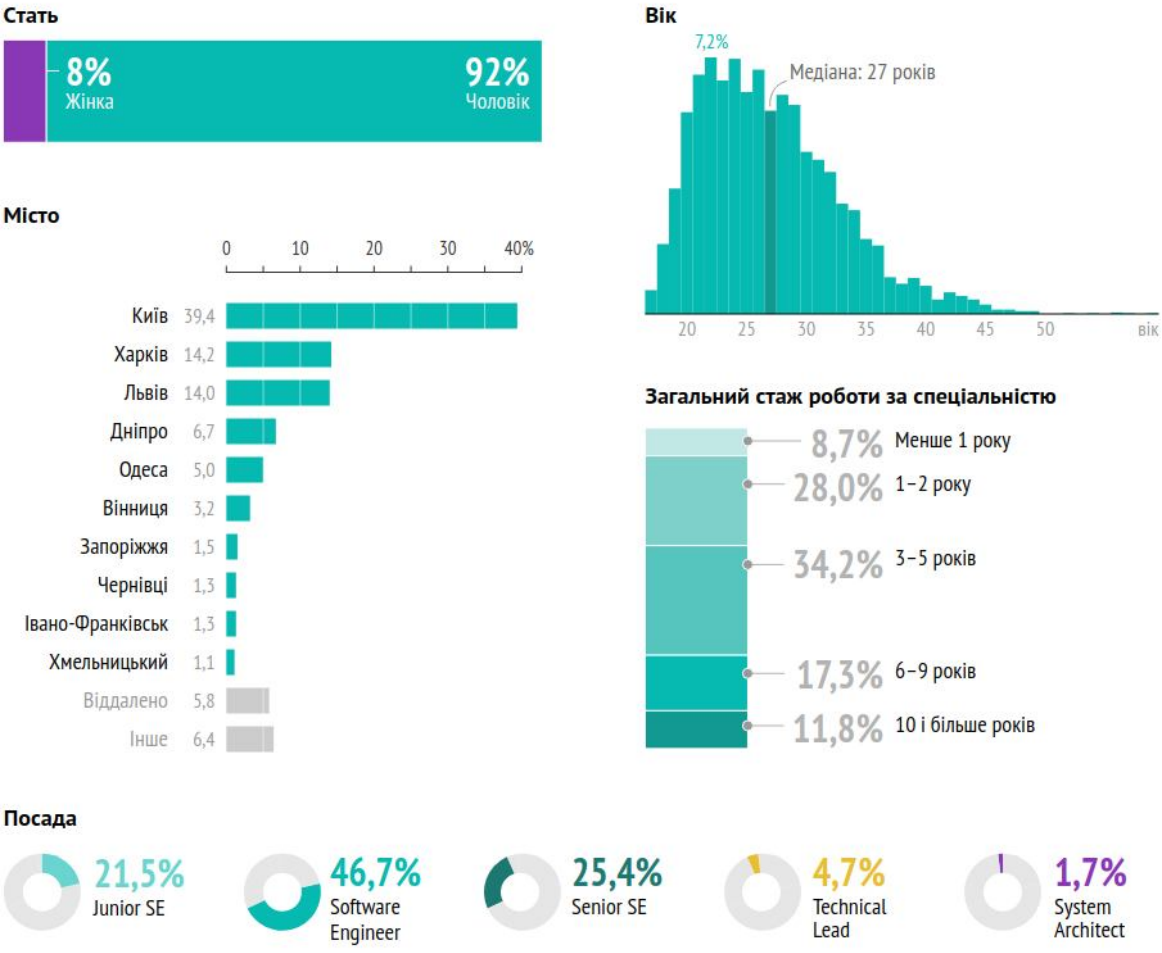


Рисунок 2.3 — Портрет учасників опитування за грудень 2019 [4]

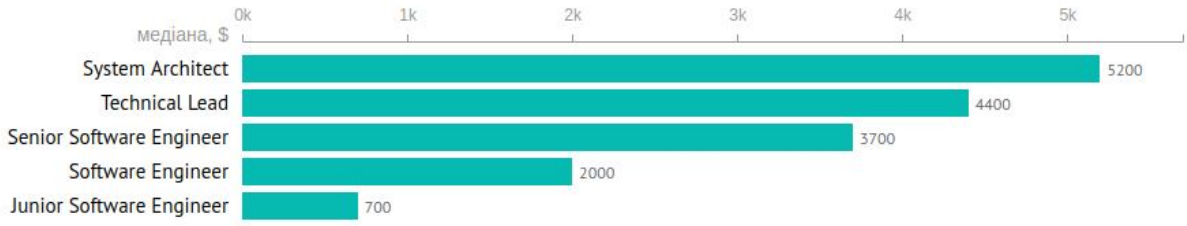


Рисунок 2.4 — Розподіл заробітної плати залежно від рівня посади [4]

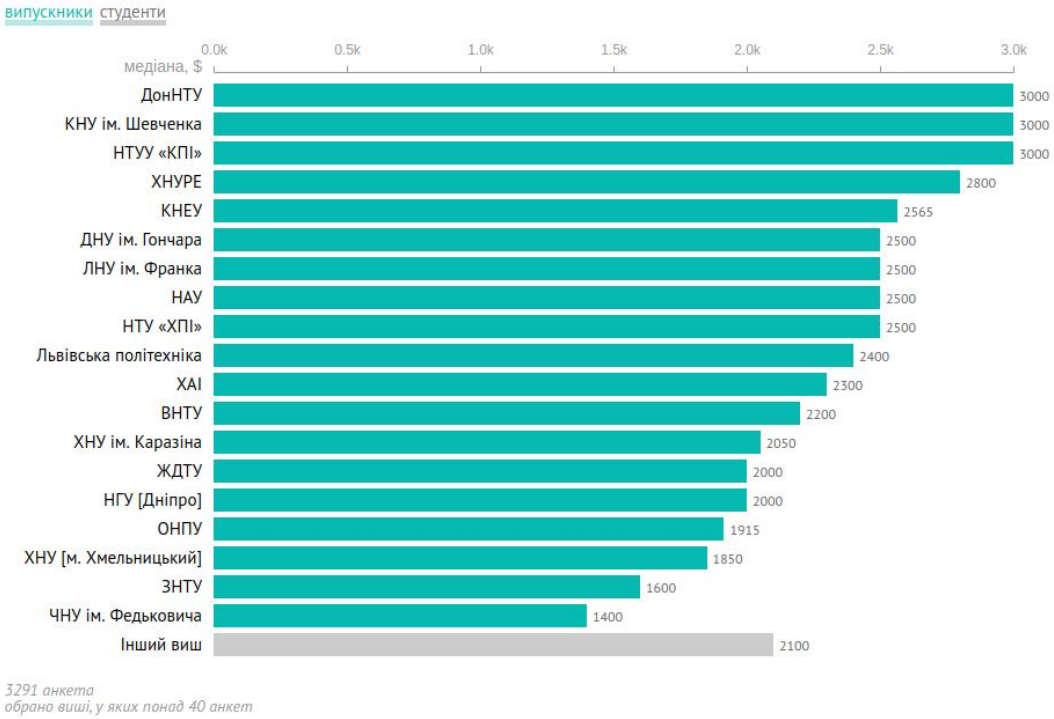


Рисунок 2.5 — Розподіл заробітної плати залежно від навчального закладу випускника [4]

Легко побачити, що як Dou, так і Djinni сильно фокусуються саме на рівні заробітної плати. Але Dou приводить більш детальний аналіз, хоч і використовує інше джерело даних і метрики. Тим не менш, такі рейтинги є малоінформативними в плані популярності технологій чи стеків технологій. Перевагою над подібними системами в розроблюваної програми має бути можливість дізнатися про зв’язок технологій, те, які технології найбільш підвищують шанс працевлаштування. Явним недоліком є відсутність будь-якої інформації щодо приросту зарплати при оволодінні новим інструментом розробки.

2.2 Незалежні рейтинги мов програмування

Такі рейтинги надають топ найпопулярніших технологій за певною методологією підрахунку. Можливі варіанти: частота пошуку в Google, кількість «Зірочок» у GitHub або ж кількість проектів на GitHub, які використовують певну мову. Такі рейтинги малоінформативні і дають можливість лише приблизно оцінити активність використання певної мови програмування розробниками. Прикладами таких рейтингів є Dou.ua, TIOBE та інші. На рисунку 2.6 подано приклад таблиці рейтингу TIOBE за травень 2020 року.

May 2020	May 2019	Change	Programming Language	Ratings	Change
1	2	^	C	17.07%	+2.82%
2	1	v	Java	16.28%	+0.28%
3	4	^	Python	9.12%	+1.29%
4	3	v	C++	6.13%	-1.97%
5	6	^	C#	4.29%	+0.30%
6	5	v	Visual Basic	4.18%	-1.01%
7	7		JavaScript	2.68%	-0.01%
8	9	^	PHP	2.49%	-0.00%
9	8	v	SQL	2.09%	-0.47%
10	21	^^	R	1.85%	+0.90%
11	18	^^	Swift	1.79%	+0.64%
12	19	^^	Go	1.27%	+0.15%
13	14	^	MATLAB	1.17%	-0.20%
14	10	vv	Assembly language	1.12%	-0.69%
15	15		Ruby	1.02%	-0.32%
16	20	^^	PL/SQL	0.99%	-0.03%
17	16	v	Classic Visual Basic	0.89%	-0.43%
18	13	vv	Perl	0.88%	-0.51%
19	28	^^	Scratch	0.83%	+0.32%
20	11	vv	Objective-C	0.80%	-0.83%

Рисунок 2.6 — Рейтинги TIOBE за травень 2020 [13]

Явною перевагою над такою системою є наявність інформації про сумісне використання технологій, а також більш інформативне в плані працевлаштування джерело даних — вакансії.

3. ЗАСОБИ РОЗРОБКИ

Під час виконання проекту було розглянуто велику кількість технологій, мов програмування та СКБД. Завдяки цьому було виявлено групу технологій, які забезпечують найбільшу гнучкість, швидкість роботи, а також простоту виконання з можливістю в майбутньому розширити систему. Також важливою є взаємодія компонентів системи, оскільки вона складається мінімум з 3-х модулів, що накладає певні обмеження [14].

3.1 Система збору та обробки даних

Основною мовою програмування для всіх модулів, крім графічного інтерфейсу, було обрано Python. Ця мова програмування створена для високорівневого опису алгоритмів загального призначення, орієнтована на підвищення продуктивності розробника [15]. Її головними перевагами стала простота і швидкість написання коду. Вона має велику кількість готових модулів і фреймворків та інтеграцій з більшістю наявних технологій, динамічну типізацію, механізм обробки винятків, підтримку багатопоточних обчислень, автоматичне керування пам’яттю, повну інтроспекцію, високорівневі структури даних [16, 17]. Мова програмування Python підтримує об’єктно-орієнтоване, структурне, імперативне, функціональне і аспектно-орієнтоване програмування. Для організації коду використовується розбиття програм на модулі, які об’єднуються в пакети [18].

Мова Python використовується в багатьох проектах для створення розширень і інтеграції додатків. Вона активно використовується для створення прототипів майбутніх програм. Також Python використовується в багатьох великих компаніях: Dropbox, Google (наприклад деякі частини Youtube і Youtube API написані мовою Python), Facebook, Instagram.

Для реалізації системи парсингу ресурсів з вакансіями було обрано фреймворк Scrapy. Це безкоштовний фреймворк для веб-краулінгу, що знаходиться у відкритому доступі та написаний мовою програмування Python. Може бути використаний для добування інформації за допомогою API або ж як веб-краулер загального застосування, хоч спочатку створювався для веб-скрейпінгу.

Архітектура проекту Scrapy побудована навколо «павуків», які по суті є автономними краулерами з заданими інструкціями. Суть таких павуків — сканувати веб-сторінки, і автоматично знаходити всі можливі посилання на ресурси, дозволені конфігурацією. При обробці сторінки можливо додати свою логіку, як, наприклад, вилучення певної інформації та її збереження для подальшої обробки. Фреймворк побудований за принципом «не повторюй себе» (англ. «do not repeat yourself»), що спрощує створення і масштабування великих проектів обходу контенту, даючи можливість розробникам повторно використовувати свій код. Фреймворк Scrapy також надає командну оболонку для веб-краулінгу, яку розробники можуть використовувати для перевірки своїх припущень про поведінку сайту.

3.2 Серверна частина

Для реалізації серверної частини підходили 2 фреймворки: Flask і Django. Перший є більш мінімалістичною версією, функції до якого можна додавати за допомогою розширень або реалізувати самостійно. Фреймворк Django у стандартній збірці включає багато модулів і функціональності, які є зайвими у деяких випадках. Фреймворк Flask розроблений для створення веб-додатків мовою програмування Python, що використовує набір інструментів Werkzeug, а також шаблонізатор Jinja2 [6]. Відноситься до категорії так званих мікрофреймворків — мінімалістичних каркасів веб-додатків, що свідомо надають лише базові функції. Пріоритетом при розробці була швидкість розробки, тому було додано розширення Flask-Restplus, що додає функціональність для типізації й контролю за вхідними даними, відповіддю серверу, автоматичного документування та створення зручного інтерфейсу для тестування API Swagger UI. При моделюванні було обрано архітектуру REST, через

те, що вона забезпечує максимальну гнучкість [8, 19]. До того ж, такий сервер можна використовувати як для мобільних додатків, так і для веб-сторінок. На рисунку 3.1 наведено API вже готового сервера.

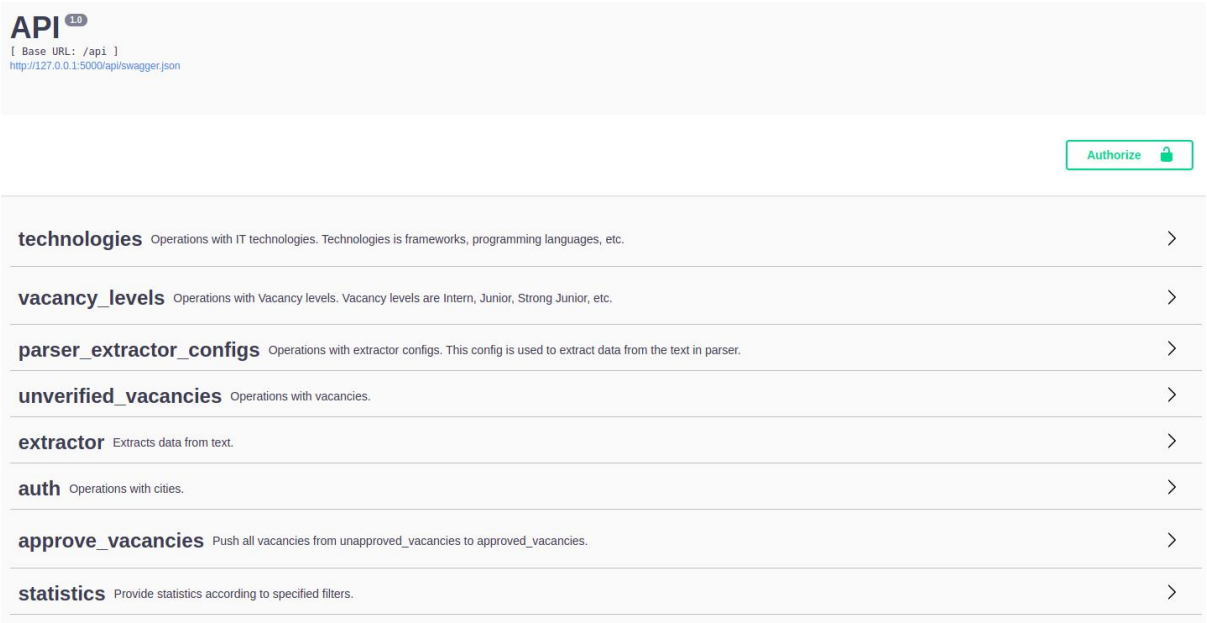


Рисунок 3.1 — Інтерфейс Swagger, перелік ресурсів, що надає REST API

Цей список сформований автоматично, згідно з ресурсами, описаними в коді. Приклад опису ресурсу technologies подано на рисунку 3.2.

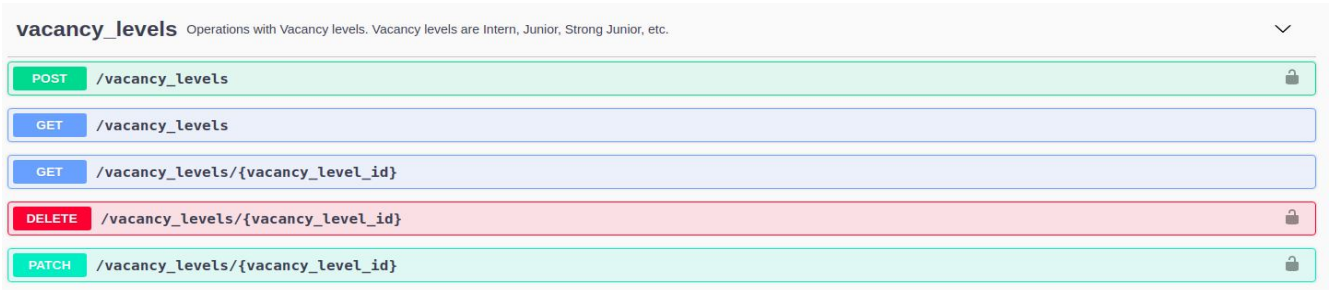


Рисунок 3.2 — Інтерфейс Swagger, перелік методів, що надає ресурс vacancy_levels

Як видно з рисунка 3.2, деякі методи є захищеними і вимагають аутентифікації. Бібліотека Swagger надає інструменти для тимчасової аутентифікації та зберігає сесію. Також надає змогу виконувати параметризовані запити, як показано на рисунку 3.3.

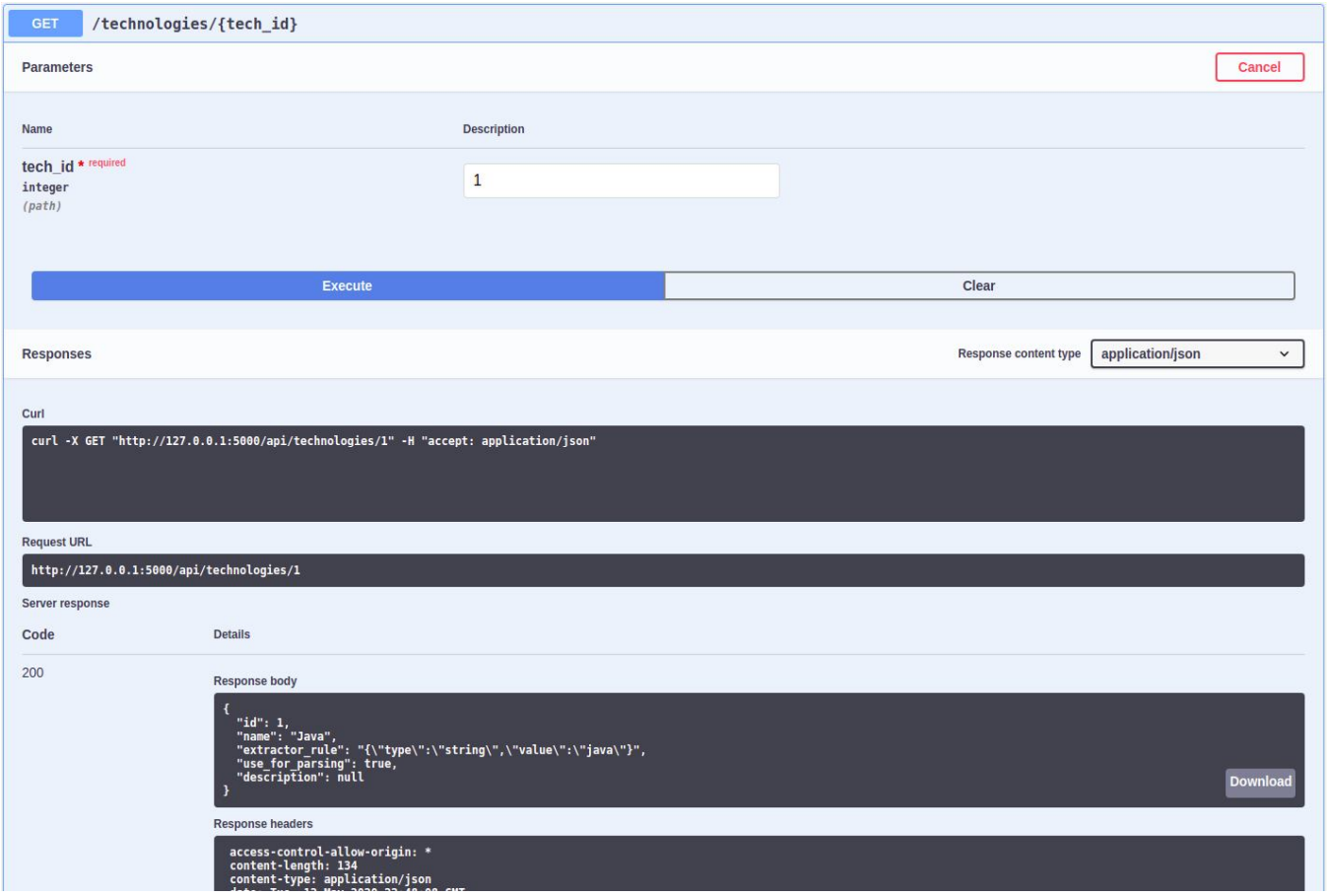


Рисунок 3.3 — Інтерфейс Swagger, параметризований запит до ресурсу technologies

Така документація дає змогу швидко зрозуміти, як сервісам, що залежать від цього сервера, необхідно взаємодіяти з системою. Також можливо відразу протестувати систему. На рисунок 3.4 показано згенеровану документацію про сутності, що приймає або віддає API.

Особливістю Restplus є те, що впроваджуються більш зручний спосіб опису ресурсів. На кожен ресурс, як наприклад «users», можна визначити 2 класи: перший, що керує операціями над колекцією, а також операції над конкретним об’єктом. Для прикладу, за додавання нового користувача буде відповідати ресурс, що керує колекцією, а за модифікацію існуючого — ресурс об’єкта. Кожен такий ресурс це клас, що може визначати HTTP-методи як методи класу. Кожен такий ресурс автоматично документується.

Також в Restplus інтегрована бібліотека Marshmallow, що відповідає за серіалізацію/дереалізацію даних від клієнта, керує структурою й типами.

Рисунок 3.4 — Інтерфейс Swagger, секція документації, яка описує сутності в системі

Автоматично також документується і створюється інтерфейс для тестування, який дає змогу відправити запит у потрібному форматі без додаткових інструментів, як Postman або cUrl.

3.3 Інтерфейс користувача

Для реалізації інтерфейсу користувача використано мультипарадигменну мову програмування JavaScript, яка підтримує об’єктно-орієнтований, імперативний і функціональний стилі. Ця мова є реалізацією стандарту ECMAScript. Як правило, використовується як вбудована мова для програмного доступу до об’єктів, додатків. Є чи не єдиною мовою, яка дає можливість будувати інтерфейси користувача у браузері з великою і активною спільнотою, розвиненими фреймворками і сформованими стандартами.

Серед великої кількості фреймворків найбільш популярними й придатними для побудови динамічного застосунку з підтримкою роботи з RESTful сервісами є Angular, React та Vue.js. Було обрано React через його мінімалістичність і більшу

пристосованість до проектів середнього розміру. Фреймворк Angular більше підходить для великих проектів, а Vue — малих.

Фреймворк React — JavaScript-бібліотека з відкритим кодом для розробки користувацьких інтерфейсів. Він розробляється і підтримується Facebook, Instagram і співтовариством окремих розробників і корпорацій. Цей фреймворк може використовуватися для розробки односторінкових сайтів і мобільних додатків. Фреймворк React дає можливість створювати інкапсульовані компоненти, які керують власним станом, а з них будувати складні інтерфейси. Оскільки логіка компонентів написана мовою JavaScript, замість шаблонів, можна передавати складні дані у додатку і зберігати стан окремо від DOM.

Було вирішено не використовувати популярну бібліотеку Redux через те, що система здебільшого не має глобального стану і може бути організована через вбудовані інструменти React-у, відомі як стани (State).

Проект Redux — бібліотека для JavaScript з відкритим вихідним кодом, призначена для управління станом застосунку. Найчастіше використовується з React або Angular для розробки клієнтської частини. Проект містить ряд інструментів, які дають можливість значно спростити передачу даних сховища через контекст. Реалізує схему видавник — підписник, де функція відображення контенту сторінки підписується на зміну стану застосунку. Зміна відбувається через інтерфейс так званих reducers, що обробляють повідомлення і змінюють певні значення в сховищі. Це дає можливість перемалювати контент сторінки при будь-якій зміні даних. Тим не менш, майже такого ж результату можливо досягти за допомогою state-механізму. Кожен компонент має свій стан, який при зміні викликає функцію, що перемальовує цей компонент на сторінці без підключення додаткових бібліотек.

Для спрощення роботи було вирішено підключити бібліотеку ReactUI, яка містить вже реалізовані компоненти у стилістиці Material Design. Ця бібліотека також задає спосіб побудови CSS, як JavaScript-об’єктів. Це дає можливість динамічно змінювати стилі під час зміни теми застосунку, наприклад, на темну. Це також дає можливість отримати більший контроль над застосунком і керувати CSS

як кодом. Також ReactUI надає колекцію вбудованих іконок, що дає можливість імпортувати менше залежностей.

Нижче наведено компоненти, які найчастіше використовувалися при побудові застосунку:

1) компоненти розміщення:

а) paper — дає можливість візуально виділити та відділити контент від фону. Часто використовується як головний компонент сторінки чи структурна одиниця;

2) поля вводу:

а) кнопки — містять вбудовані анімації, схожі до тих, які широко застосовують при мобільній розробці;

б) текстовое поле — зручне текстовое поле, подібне до тих, які використовує компанія Google. Головними особливостями є виділення та анімації. До того ж, такі поля легко впізнавані;

в) поле вибору (Select) дає можливість обрати один чи кілька варіантів серед запропонованих;

г) поле автозаповнення схоже на поле вибору, але також дає можливість фільтрувати варіанти за допомогою звичайного текстового вводу. Можна користуватися ним як текстовим, так і як полем вибору. Дуже корисний при великій кількості варіантів вибору;

3) зворотній зв’язок:

а) інформаційні повідомлення (SnackBar) — має кілька видів повідомлень, використовується для відображення статусу виконання певної операції. Наприклад, може відображати статус «Успішно» чи «Помилка» з повідомленням. Особливістю є те, що зникає автоматично через заданий проміжок часу і не заважає користувачеві;

4) зображення даних:

а) іконки;

б) таблиці.

3.4 База даних

Для зберігання даних було обрано 2 бази даних: MongoDB, а також SQLite. База даних MongoDB — документоорієнтова система керування базами даних з відкритим вихідним кодом, яка не потребує опису схеми таблиць [20]. Класифікована як NoSQL, використовує JSON-подібні документи і схему бази даних. Вона написана мовою програмування C++. Система може працювати з набором реплік, тобто містити дві або більше копії даних на різних вузлах. Кожен екземпляр набору реплік може в будь-який момент виступати в ролі як основної, так і допоміжної репліки. Всі операції запису і читання за замовчуванням здійснюються з основної репліки. Допоміжні репліки підтримують в актуальному стані копії даних. Тобто реалізована master-slave реплікація [21]. У разі, коли основна репліка дає збій, набір реплік проводить вибір, яка з реплік має стати основною. Другорядні репліки можуть додатково бути джерелом для операцій читання. Така система забезпечує високу швидкість для операцій читання-запису і, хоч і не призначена для такого, надає широкі можливості для виконання аналізу та операцій агрегації й фільтрації. Ця база була обрана для зберігання вакансій, оскільки структура даних містить багато масивів, які вимагають побудови складних зв’язків у системах з реляційною парадигмою. Використання Mongo пришвидшує розробку в кілька разів, надає операції фільтрації, що було б складно реалізовувати в SQL-базах. Також ця база надійно працює з великими обсягами даних, що проблемно для більшості реляційних систем [22]. Тим не менш, така база не може забезпечити всі потреби, в основному через те, що у ній складно підтримувати консистентність даних і багато перевірок перекладається на користувача [23]. Тому було вирішено використовувати також базу даних SQLite, яка без проблем може бути замінена на PostgreSQL в майбутньому при розвитку системи.

База даних SQLite зберігає дані про користувачів для аутентифікації, конфігурації, які використовуються системою обробки даних. Ця база повністю підтримує мову SQL (структурні зміни, маніпуляції з даними й транзакції). Її

перевага на даному етапі — простота інтеграції з Python, швидке налаштування і те, що всі дані зберігаються у файлі, який задається користувачем. Для простішої реалізації було використано розширення для Flask, SQLAlchemy, що реалізує ORM-технологію програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних». Як результат, структура бази створюється і описується у коді, а всі операції щодо версіювання структури бере на себе інше розширення — Flask Migrate, яке реалізує підхід схожий на Git, тільки зі структурою бази даних [24]. Це дає можливість запобігти проблемам з різними версіями коду і бази. Будь-які зміни структури даних у коді будуть автоматично вирахувані й занесені в базу при наступній міграції. Це позбавляє програміста необхідності писати запити для зміни структури. Цікавою також є можливість однією командою відмінити зміни.

3.5 Розгортання та обслуговування сервісів

Для забезпечення узгодженої роботи й незалежності від інфраструктури, готові сервіси було згорнуто у контейнери Docker та об'єднано у Docker-compose. Система Docker — інструментарій для управління ізольованими Linux-контейнерами. Він надає високорівневий API, який дає можливість керувати контейнерами на рівні ізоляції окремих процесів. Зокрема, Docker дає можливість, не переймаючись вмістом контейнера, запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів. Система Docker дає можливість налаштовувати мережу для контейнерів, відкривати деякі порти, що дасть можливість звертатися до них ззовні. Наприклад, веб-застосунок, запущений у Docker-контейнерах, має відкрити ті порти, які прослуховує сервер всередині. Так, для npm serve, як правило, використовується порт 3000, а це означає, що саме його треба відкрити зсередини. Також треба вказати, який порт буде доступний ззовні контейнера. Для прикладу, можливо всі запити на локальній машині з порту 80 перенаправити на 3000 порт контейнера.

Важливим також є здатність створювати volumes — функція, яка дає можливість використовувати том файлової системи хосту в середині контейнерів. Це дає можливість зберігати деякі дані, які виникають при роботі контейнера, як, наприклад, файли бази даних.

Технологія Docker-Compose — це інструмент для визначення й запуску багатоконтейнерних програм Docker. За допомогою Compose можна використовувати файл YAML для налаштування служб своєї програми. Потім за допомогою однієї команди створюються й запускаються всі служби зі своєї конфігурації.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Система складається з 3-х сервісів і коду, винесеного в окремий модуль. Для кращого розуміння роботи системи необхідно розглянути схему взаємодії компонентів (рисунок 4.1).

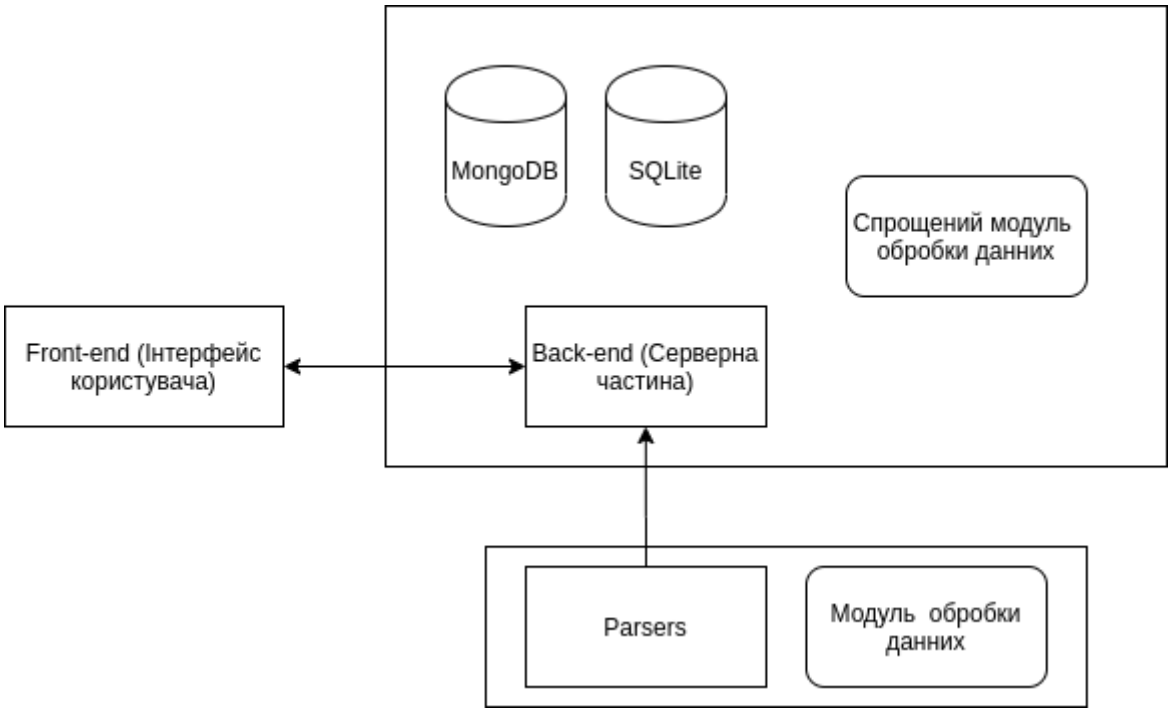


Рисунок 4.1 — Структура взаємодії сервісів системи

Парсери запускаються кожен день і опрацьовують всі вакансії, доступні на даний момент. Як правило, ресурси надають лише вакансії, які були активні не більше ніж місяць тому. Кожен парсер має надати такі дані:

1) id_external — рядок, що містить унікальний ідентифікатор з системи, з якої вакансія була взята. Обов’язкове поле;

2) data_source — рядок, ідентифікує унікальну назву сервісу-джерела вакансії. Обов’язкове поле;

3) url — шлях до вакансії;

4) title_raw — заголовок вакансії в неопрацьованному вигляді;

5) publication_date — дата публікації вакансії (часто недоступне поле);

6) date_parsed — дата, коли вакансія була зпарсена;

7) required_technologies_raw — текст вакансії (або ж певна його частина, яка містить інформацію про технології, які необхідно знати працівникові. Часто буває дуже проблематичним виділити необхідну частину тексту, тому до аналізу зберігається весь опис вакансії).

Наступним кроком буде опрацювання даних так званими екстракторами. Це головна частина модуля з обробки даних, що залежить від конфігурації, яка надається у форматі JSON і здатна виокремлювати ключові слова. Таких екстракторів у системі 2: перший виокремлює технології, такі як Java, Java Script та інші, а другий — рівень вакансії. При детальному аналізі було виявлено такі основні рівні вакансії:

1) початківець, інтерн —Trainee;

2) молодший спеціаліст — Junior;

3) висококваліфікований молодший спеціаліст — Strong Junior;

4) спеціаліст — Middle (Regular);

5) висококваліфікований спеціаліст — Strong Middle;

6) старший спеціаліст — Senior;

7) висококваліфікований старший спеціаліст — Strong Senior;

8) лідер команди чи технологічний лідер — Lead;

9) архітектор — Architect;

Технології і рівні вакансії у більшості випадків описуються англійською мовою, тому в системі підтримується тільки вона.

Екстрактори даних додають такі поля:

1) core_technologies — технології, наявні у заголовку;

2) vacancy_level — рівень вакансії, вказаний у заголовку;

3) required_technologies — технології, знайдені в описі вакансії.

Усі зібрані вакансії зберігаються до колекції unverified_vacancies у MongoDB. Користувачі не мають доступу до цієї колекції. Таке рішення дає можливість провести певний аналіз чи агрегацію на даних. Наприклад, можна додати функціонал, який дає можливість переглядати вакансії і виправляти інформацію в

ручному режимі. Це також дасть можливість допрацьовувати конфігурації системи. Для роботи екстрактора даних необхідні закодовані у JSON правила, які він може отримати від серверної частини. Також у майбутньому можливо буде залучити оператора до перевірки і виділення більшої кількості інформації з вакансій. Так, наприклад, є досить проблемним програмі виділити не тільки технології, а й досвід, необхідний інженеру з ними. Складним також є виділення того, яка технологія є обов'язковою, а яка стане перевагою і є лише бажаною. Слід враховувати, що оператор не здатний обробляти дані так швидко, як комп'ютер, тому логіку, яка вимагає швидкої реакції на появу тих чи інших вакансій треба переносити на цю колекцію, при цьому, колекція з якою буде працювати оператор вже має бути очищена від дублікатів, що може вимагати значних ресурсів у майбутньому.

4.1 Сервіс обробки даних

Головним класом цього сервісу є клас Extractor, який реалізує єдиний метод extract(String): ExtractorResponse. Під час ініціалізації конфігурації, надані REST-сервером, десеріалізуються і задають поведінку сервісу. У даному випадку десеріалізацією є перетворення конфігурації у форматі JSON на Python-структури. За це відповідає окремий клас RuleBuilder, який інкапсулює логіку побудови всередині себе, що дасть змогу швидко замінити її в разі необхідності, не змінюючи інший код. Конфігурація містить багато параметрів, які мають бути десеріалізовані. Для прикладу, можна задати правило, що обов'язково має бути задоволене (наприклад, в тексті повинно бути слово «english»), інакше буде повертатися порожня відповідь. Також можливо встановити максимальну відстань у словах від обов'язкового правила, на якій мають позміщуватися всі слова, що задовольняють правила, або ж будуть ігноровані. Також можливе значення за замовчуванням.

На рисунку 4.2 подано діаграму класів для правил, які використовуються екстрактором. Екстрактор маніпулює словником (структурою, яка зберігає дані типу ключ-значення), де ключем є ідентифікатор у базі. Це, наприклад, ідентифікатор

певної технології, а значенням є клас Rule, що відповідає за перевірку, чи відповідає текст необхідній умові.

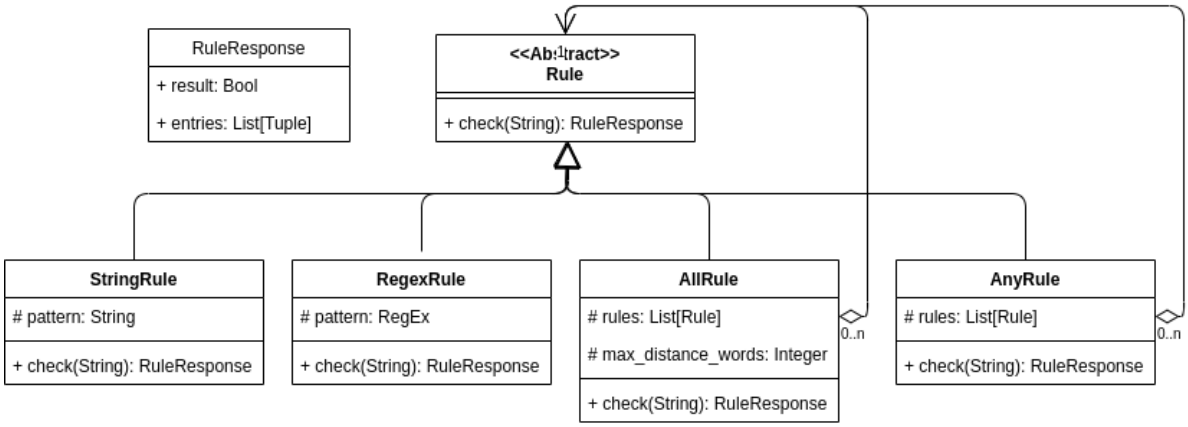


Рисунок 4.2 — UML-діаграма класів для Rule

Як видно з діаграми, класи AllRule та AnyRule реалізують шаблон Компонувальник [19]. Це дає змогу інкапсулювати складну логіку і працювати з нею як з простим правилом. Клас StringRule просто перевіряє, чи наявне слово в тексті, ігноруючи регістр. Так само і RegexRule, тільки Regex шукає не просто слово, а перевіряє, чи наявні збіги з регулярним виразом. Клас AllRule — комплексне правило, яке перевіряє, щоб усі вкладені правила були виконані. Координатами збігу будуть крайні координати збігів з простими правилами, які формують AllRule. Клас AnyRule — перевіряє, щоб був хоч один збіг.

Клас Extractor побудований так, щоб знаходити всі збіги і допускати якнайменше колізій. Нижче подано алгоритм роботи екстрактора:

1) приведення тексту до нижнього регістру;

2) якщо наявне обов’язкове правило:

а) якщо правило не задане — пропускаємо;

б) якщо правило задане:

(1) якщо текст задовольняє правило, записуємо координати всіх збігів у вигляді (координата початку, координата кінця);

3) для всіх правил формуємо список з об’єктів, які містять (ключ правила, координати збігу);

4) фільтруємо всі збіги, які є частиною більшого правила;

5) якщо задана максимальна відстань від обов'язкового правила, то фільтруються всі збіги, розміщені занадто далеко.

Також у коді є кілька перевірок, які логують статус виконання і дадуть змогу знаходити можливі помилки оператору.

Важливим є алгоритм роботи правила AllRule, оскільки, як видно з попереднього алгоритму, збіг має 2 координати в тексті: координату початку і кінця. А отже, якщо обрати найбільше правило у тексті, воно може містити в собі кілька менших, що зламає логіку, оскільки код в Extractor просто візьме його й видалить всі інші збіги.

Нижче подано алгоритм роботи AllRule:

1) правило знаходить всі збіги вкладених правил;

2) для всіх збігів знаходяться комбінації. Варто зазначити, що якщо AllRule містить 3 вкладених правила, то знаходяться комбінації по три збіги різних правил. Якщо 2 вкладених правила — попарні комбінації;

3) видаляються правила, які містять у собі інші правила (тобто лишаються лише найменші збіги або, іншими словами, залишаються лише ті збіги, які розміщуються найближче, не перетинаються з іншими правилами і формують повний збіг AllRule).

На рисунку 4.3 подано приклад, у якому екстрактор знайде технології Java і JavaScript. Тут конфігурація задана двома правилами: String(Java) і All(String(Java), String(Script)). Завдяки тому, що AllRule залишає найменші правила, що не перетинаються, то буде обрано серед двох варіантів JavaScript саме менший. Помилковими варіантами можуть бути: тільки JavaScript, 2 рази Java.

У попередньому прикладі всі зайві збіги відфільтрував клас AllRule, а на рисунку 4.4 подано приклад для ситуації, коли правила задано такою конфігурацією: String(Java), String(Java Script).

У такому випадку фільтрувати код буде вже сам Extractor. Як відомо з наведеного вище алгоритму, він обирає більші за розміром правила серед тих, що перетинаються, тобто в цьому випадку будуть обрані Java і JavaScript.

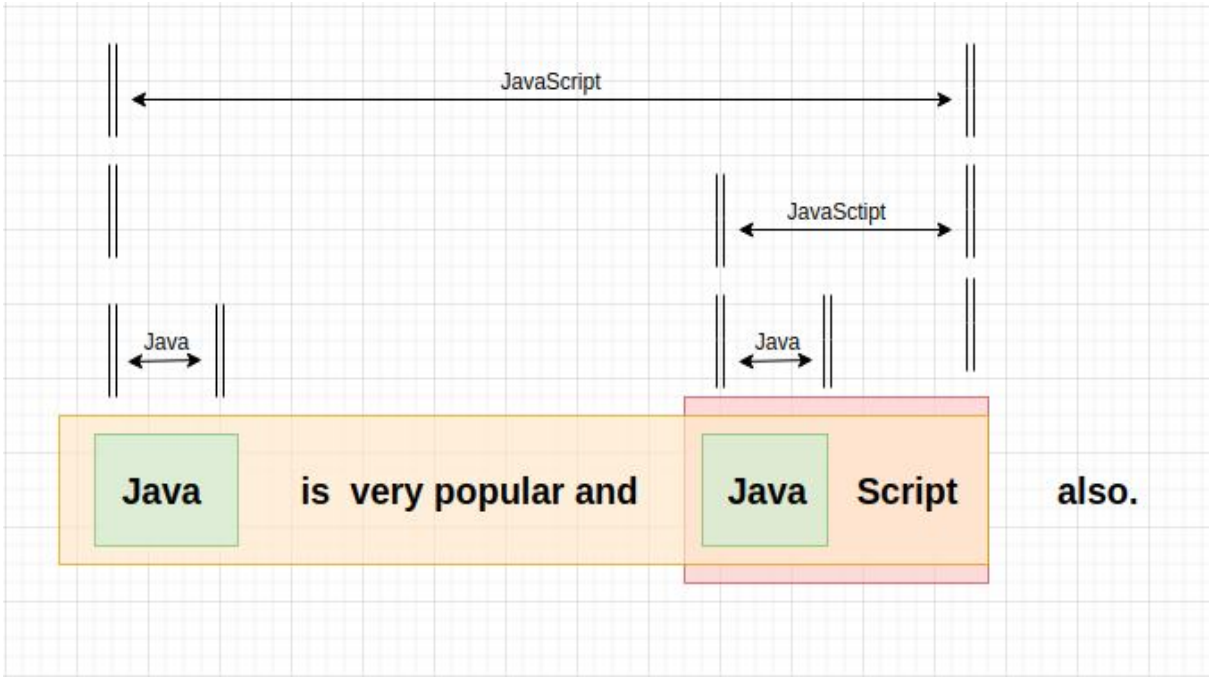


Рисунок 4.3 — Складна ситуація — AllRule

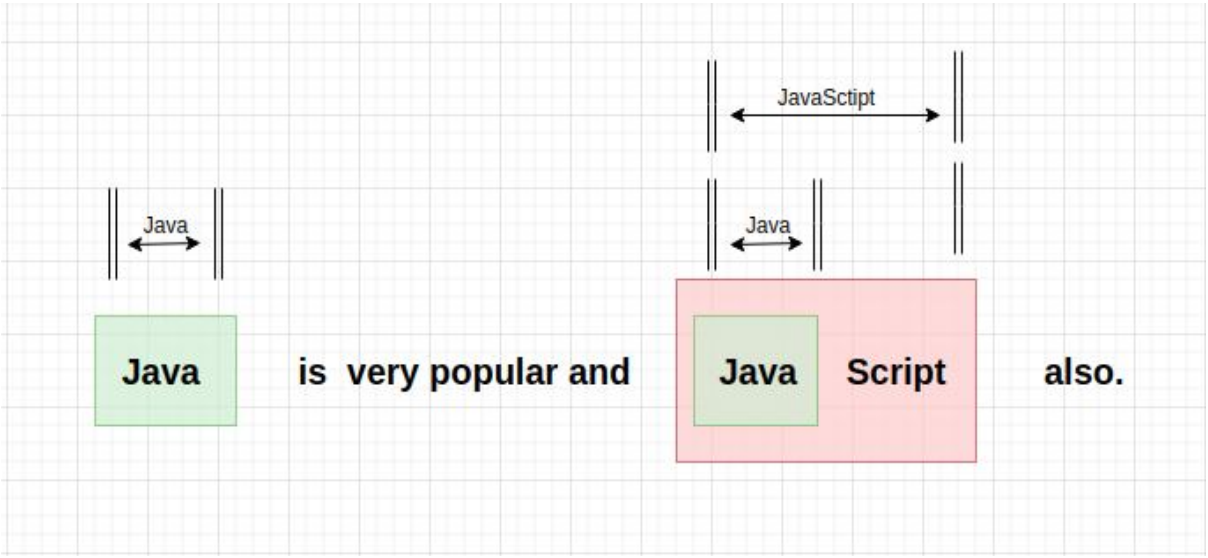


Рисунок 4.4 — Складна ситуація StringRule

Для забезпечення стабільності коду багато подібних ситуацій було прописано у вигляді Unit-тестів [25]. Це, як правило, автоматизовані тести, написані й запущені розробниками програмного забезпечення, щоб переконатися, що розділ програми (відомий як «Unit») відповідає його дизайну й веде себе за призначенням. В об’єктно-орієнтованому програмуванні Unit часто є цілим інтерфейсом, наприклад, класом, але може бути індивідуальним методом. Написавши тести спочатку для

найменших тестових одиниць, потім складної поведінки між ними, можна скласти комплексні тести для складних застосувань. Тести написані за допомогою бібліотеки Unittest, яка реалізує багато необхідного функціоналу, як, наприклад, методи перевірки правильності умови й оточення запуску. Покриття коду тестувалося за допомогою coverage.py.

Тестами покрито 87% рядків коду і 83% файлів. Кожен набір тестів відповідає бізнес-вимогам, які ставляться для коду. Всього таких тестів 34. Модуль був повністю реалізований за допомогою стандартної бібліотеки Python [26].

4.2 Сервіс збору даних

У рамках реалізації цього сервісу було реалізовано 2 «Павуки», кожен з яких створений, щоб парсити з одного і тільки одного веб-ресурсу, оскільки структура кожного веб-сайту неповторна. Система працює з сайтами Epam і Djinni, які надають публічну інформацію про вакансії. Оброблюється як статичний HTML, так і JSON відповіді з REST API. Також було налаштовано 2 «конвеєри» даних: перший використовує сервіс обробки даних для того, щоб додати необхідні колонки до даних, а другий зберігає опрацьовану вакансію. Весь інший функціонал бере на себе фреймворк Scrapy. Фреймворк не дозволяє зберігати інформацію вбудованими методами. Його головне завдання — обійти всі можливі сторінки ресурсу з автоматичним пошуком нових. Тим не менш, він надає можливість паралельно обробити кожну сторінку й дані за допомогою побудованих конвеєрів. Таких конвеєрів може бути кілька і викликаються вони згідно з пріоритетами, що вказуються у конфігурації. Наразі у системі наявно три таких шляхи даних:

1) конвеєр встановлення значень за замовчуванням та екстракцією даних — у багатьох випадках необхідні дані містяться в тих самих полях, тому логіку з їхнього опрацювання можна винести з класу «Павука» і зробити спільною для всіх. Тим не менше, якщо якісь ресурси міститимуть особливу структуру, її можна буде опрацювати прямо в коді «Павука»;

2) конвеєр контролю якості — має перевіряти критичні аспекти, як, наприклад, наявність ідентифікатора, який використовується для пошуку дублікатів. Цей сервіс може видаляти певні вакансії запобігаючи їхній подальшій обробці й зберіганню. Також тут можливо розмістити логування;

3) конвеєр збереження.

Логіка збереження інкапсульована у класі й задекларована в інтерфейсі. Це дає змогу підмінити реалізацію та зберігати дані в різних базах чи форматах файлів без зміни коду.

Також було реалізовано логіку логування, здатну зберігати інформацію про помилки при виконанні коду, а також контролювати якість зібраної інформації. Цей код наразі не є приєднаним до бази, але вже має всі необхідні інтеграції і в майбутньому дасть змогу реалізувати додаткові сторінки з аналізом проблемних місць у системі.

4.3 Серверна частина

Для забезпечення взаємодії всіх компонентів було прийнято рішення реалізувати RESTful API-сервер. За основу було взято фреймворк Flask та Flask-Restplus. Архітектуру застосунку повністю задає Flask-Restplus. У системі наявні 2 типи ресурсів: колекція (або лист) і сутність.

Для реалізації нової сутності було вирішено використовувати Namespaces, який ізолює логіку в окремий модуль, що легко можна масштабувати, створювати версії, перевикористати як окремий сервіс [27]. Як правило, Namespaces мають у собі по одній колекції та сутності, що ідентифікуються відповідним URL. Для прикладу наведено зображення автоматично створеної інтерактивної документації (рисунок 4.5):

Як видно з рисунка 4.5, в системі наявні 8 ресурсів. Для прикладу, technologies містить колекцію технологій (дії з багатьма технологіями) та сутність «технологія» (дії над конкретною технологією, що ідентифікується за унікальним ID (рисунок 4.6)).

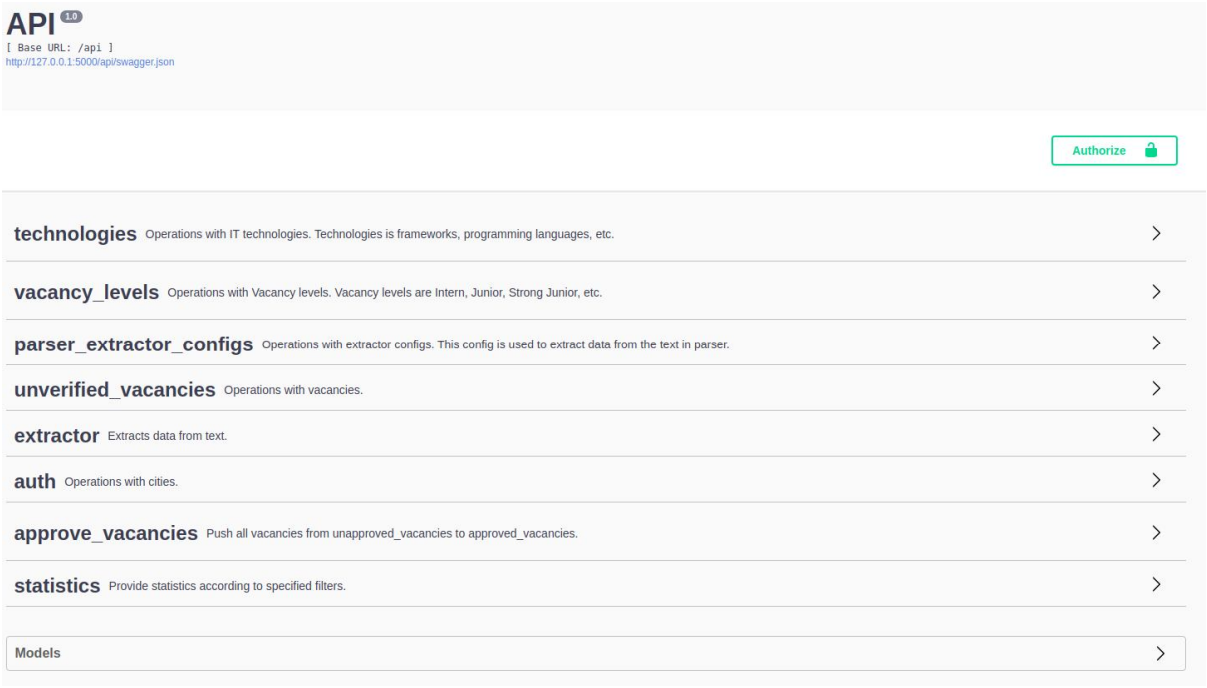


Рисунок 4.5 — Зображення всіх восьми Namespaces в системі

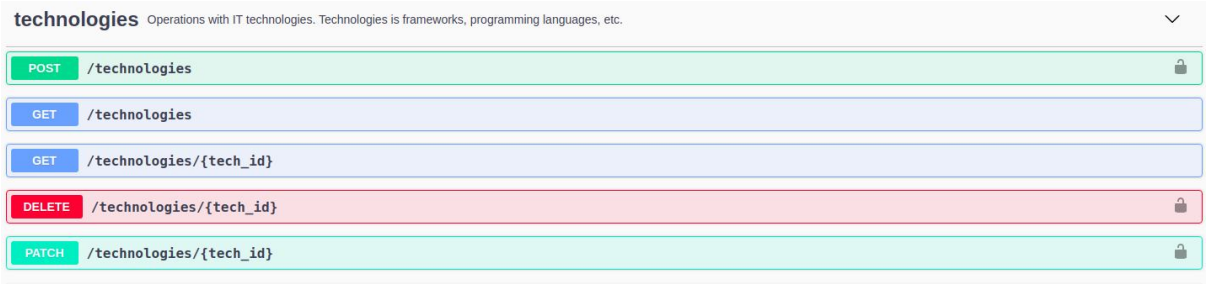


Рисунок 4.6 — Перелік доступних операцій над технологіями

Як видно з рисунка 4.6, доступними є п’ять операцій над технологіями:

1) отримати всі;

2) створити нову;

3) отримати за ID;

4) оновити певну технологію за ID;

5) видалити певну технологію за ID.

Також можна побачити, що деякі методи вимагають аутентифікації (на рисунку 4.6 видно замок). Система контролю доступу й аутентифікації була реалізована самостійно і є імплементацією oAuth2. В auth Namspace є три основні методи — login, refresh і logout. Метод login приймає логін і пароль користувача,

перевіряє їхню правильність у базі даних, після чого утворює 2 токени, які є закодованими повідомленнями, що можуть бути декодовані тільки, якщо є секретний ключ, яким токен був закодований. Один токен — accessToken використовується для доступу до ресурсів і є актуальним лише 15 хвилин. Другий токен може бути використаний 2 дні. Його використовують для отримання нової пари токенів, коли accessToken псується. Це дає змогу вберегти себе від ситуації, коли зловмисник вкрав accessToken, оскільки він може ним користуватися лише 15 хвилин. Також цей підхід дає можливість зберігати дані в теконі і не звертатися зайвий раз до бази, щоб дізнатися, чи є в користувача доступ до певного ресурсу.

В accessToken закодовано таку інформацію:

1) exp — дата, до якої токен вважається валідним;

2) user_id;

3) role_level — int, звичайний користувач має значення 1, адміністратор — 10. Таке велике вікно створено для того, що була можливість додати ще ролей;

4) email.

В refreshToken зберігається:

1) exp — дата, до якої токен вважається валідним;

2) token_id.

Коли користувач викликає refresh, то з бази видаляється старий refreshToken і замінюється на новий. Якщо токен в базі не знайдено, — його вже використали. У такому випадку потрібно знову вводити логін і пароль. Усі токени кодуються за алгоритмом HS256 (HMAC-SHA256), що вимагає лише один секретний ключ.

Крім ресурсів, у системі також наявні спеціальні функції. Перша функція — approve_vacancies, яка фільтрує всі вакансії з колекції unverified_vacancies, фільтрує дублікати (вакансії, вже наявні в системі), а також вакансії, які були неправильно розпарсені. Прикладом такого може бути вакансія, яка не має жодної технології або ж має більше 15 технологій. Також помилковими є вакансії, у яких не має рівня вакансії, після чого дані записуються в колекцію vacancies, яка вже використовується для побудови статистики. Колекція unverified_vacancies відчищається.

Також наявна функція extract, яка використовується для тестування певного конфігу для екстрактора. На вхід вона приймає конфігурацію й текст. На виході віддає масив з даними, які змогла виокремити з тексту.

Для побудови статистики використовується функція statistics. Вона приймає на вхід технології, а також рівні вакансії, за якими треба створити статистику. Обов'язково мати принаймні одну технологію, інакше буде повернено порожню відповідь. Код вибирає з бази всі вакансії, що містять хоча б одну технологію зазначену в фільтрі. Це зроблено для того, щоб мати можливість аналізувати також технології, які використовуються у купі з обраними.

При моделюванні системи було прийнято рішення створити і підтримувати дві бази даних. База MongoDB забезпечує високу швидкість на зчитування й запис і надає широкі можливості для аналізу. До того ж добре підлаштовується під великі обсяги даних. Проектована система планувалася як така, що буде змінюватися і розвиватися, що накладає побічні витрати на підтримку реляційної моделі з чіткою схемою. Також у системі планувалася можливість додати логування з нечіткою структурою, що також вимагає рішень, здатних досить гнучко опрацьовувати данні. Набагато легше мати вбудований масив в об'єкті, а не будувати відношення один до одного.

Робота з MongoDB вимагає денормалізації, тобто процесу перетворення даних, протилежного до нормалізації [28]. Такий процес часто призводить до компромісів, а також перекладає необхідність контролю консистентності на користувача [29]. Наприклад, якщо адміністратор вирішить оновити назву технології, то доведеться також робити додаткові запити для всіх вакансій, у яких присутня ця технологія і перейменовувати їх.

Тут можливі колізії, за якими деякі технології залишаться без змін, що є помилкою. До того ж, важко слідкувати за тим, щоб не видалити технологію, яка вже наявна в вакансіях. Для розв'язання цих проблем повністю підійшов комплекс з SQLite, який дав можливість зберігати стійкі та чутливі дані в реляційному середовищі, а все інше в MongoDB. На рисунку 4.7 подано структуру реляційної бази.

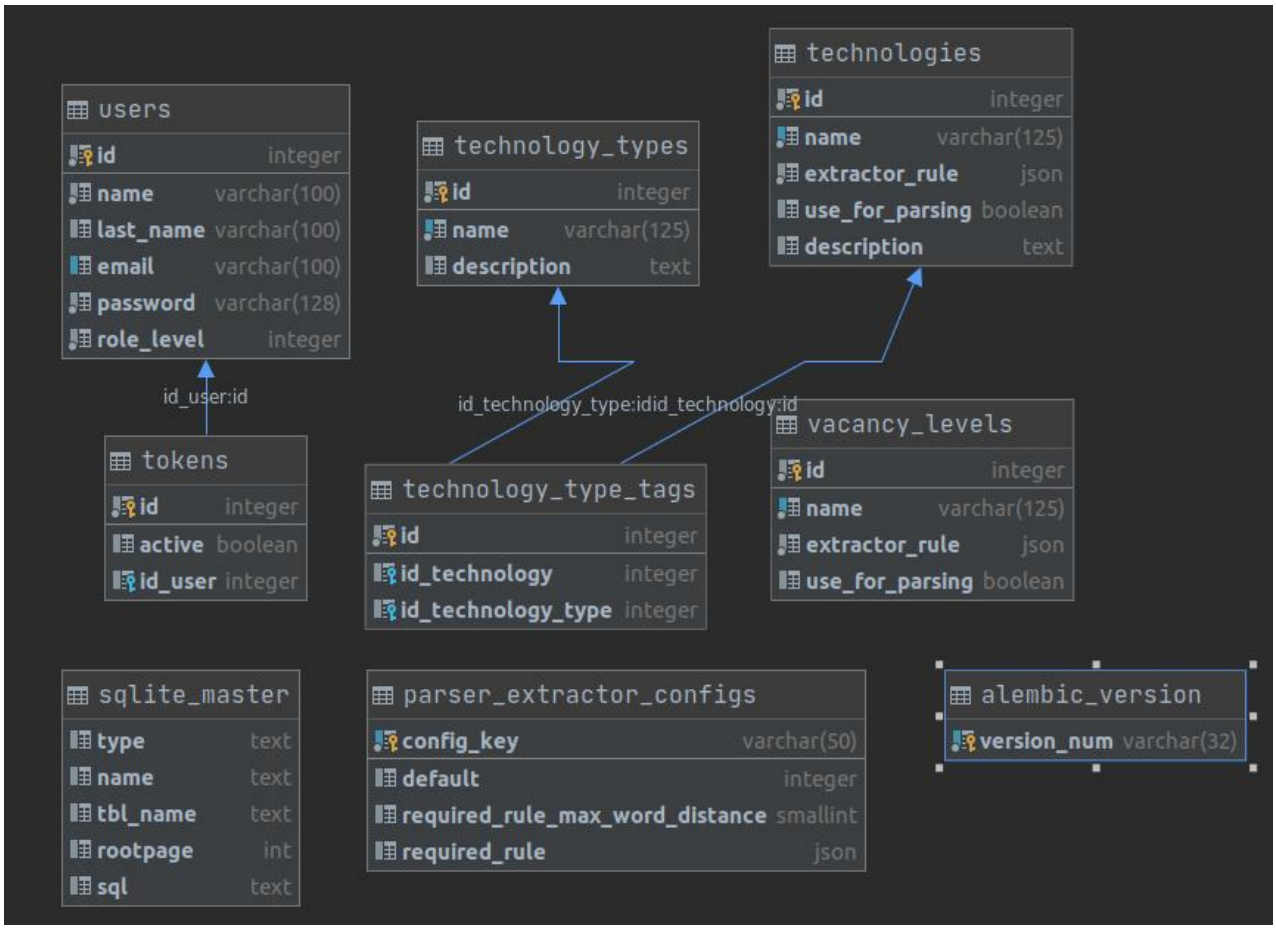


Рисунок 4.7 — Структура таблиць бази SQLite

Тут наявні дві службові таблиці: Sqlite_master — таблиця з мета-інформацією, яка була створена самою СУБД, а також alembic_version, яка дає можливість версіювати структуру таблиць бази даних, оновлюватися до нової, а також повертати зміни назад. Усе, що містить alembic_version — це ідентифікатор версії бази, яка зараз використовується. Сам опис розміщується у файлах проекту. Розширення Flask-Migrate створює окрему папку, де зберігаються автоматично згенеровані файли python, які містять два методи:

1) upgrade — відповідає за створення чи оновлення бази;

2) downgrade — дає змогу повернутися до попереднього стану.

Комунікація з SQLite відбувається через бібліотеку sqlalchemy. Це дає можливість описати структуру таблиць у коді, без написання SQL. Головною перевагою такого підходу є можливість перейти на базу PostgreSQL майже без зміни коду.

4.4 Інтерфейс користувача

Інтерфейс було реалізовано за допомогою фреймворку React. Він націлений на побудову інтерфейсів за принципом єдиної відповідальності. Тобто структурні блоки-компоненти в системі мають відповідати за якомога меншу кількість логіки. Тим не менш, компонент може бути зіставним. Хорошим підходом в React є побудова так званих «розумних» і «дурних» компонентів. Перші відповідають за бізнес-логіку, звернення до сервера, підрахунки тощо. Другі лише відображають якусь частину інтерфейсу.

Було використано утиліту create-react-app, яка створює базову структуру проекту, тим не менш, вона не диктує як будувати застосунок. Було обрано структуру, подану на рисунку 4.8.

Папка public містить усі статичні файли, як наприклад іконки, що будуть використані при побудові інтерфейсу. Папка src містить саме код та основну бізнес-логіку. Файл Index.js — точка входу в програму, а також головний html-файл, який буде відображатися в браузері. Він ініціює бібліотеки, а також містить головний компонент App.js, який є точкою входу для всього JavaScript коду. Тут також наявні тексти, які перевіряють те, що код компілюється, і те, що CSS-файли є спільними для всієї системи.

Було прийнято рішення для економії часу використовувати фреймворк з уже готовими компонентами у стилі Material, дуже схожий на той, що використовує Google.

Його особливістю є підхід до опису стилів як коду. У випадку, коли компоненти досить великі, такий підхід є невиправданим, тим не менше, правила написання гарного коду в React стверджують, що компоненти повинні бути порівняно маленькими, бо інакше можуть бути декомпозовані на кілька менших.

Застосунок дотримується принципів SPA (single page applications), що означає, що він не повинен перезавантажуватися, аби відобразити новий контент, навіть, якщо це нова сторінка. Тим не менш, історія переходів за посиланнями має зберігатися браузером. Такий підхід значно пришвидшує роботу сайту, оскільки не

потрібно завантажувати з сервера HTML, стилі й бібліотеки на кожен новий запит, а лише дані, які треба відобразити. Це також робить застосунок набагато інтерактивнішим.

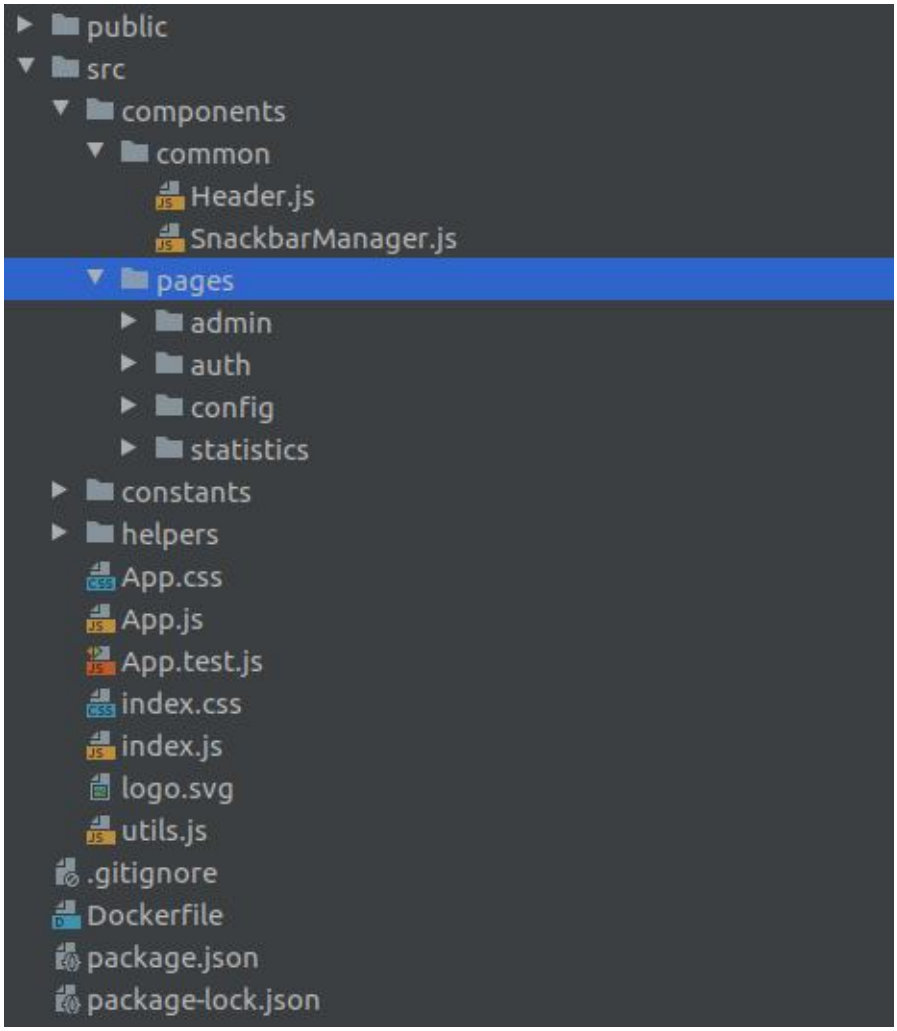


Рисунок 4.8 — Структура проекту

Зрозуміло, що для реалізації цього необхідно змінити типову поведінку посилань у системі. Те саме стосується і роутингу. Було застосовано спеціальний роутинг, який залежно від шляху в URL компілює певний компонент. Такі компоненти в системі зберігаються в папці components/pages.

Для спілкування з користувачем необхідно чітко давати повідомлення про помилки, а також статус операцій. Для цього було розроблено компонент, який за допомогою функцій зворотного виклику, а також стандартних станів React дає змогу відображати повідомлення, що зникають через 3 секунди, різних рівнів: інформація,

успіх, попередження та помилка. Кожен має свій колір для кращого розуміння. Перевагою такого рішення є можливість налаштовувати зовнішній вигляд повідомлень на відміну від стандартного функціоналу браузера, хоч користуватися компонентом так само зручно. Для реалізації функціоналу було застосовано шаблон Одинак (Singleton) [19], оскільки компонент має зберігати спільні посилання на всі екземпляри.

Сервер вимагає аутотентифікації для доступу до певних ресурсів, тому весь зв’язок повинен відбуватися через спеціальний компонент, що також реалізує Singleton і додає необхідні дані в заголовки запитів, опрацьовує коди результату, автоматично оновлює токен, а також перенаправляє на сторінку логіну за потреби.

Для відображення графіків було використано бібліотеку Charts.js. Проблемою було те, що вона не була здатна працювати з React, тому довелося її портувати і створювати новий компонент Chart.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для забезпечення стабільної роботи створеної програмної системи потрібно дотримуватися основних вимог при її інсталяції, а також рекомендацій щодо використання.

5.1 Інсталяція системи

Система використовує інструменти docker та docker-compose, завдяки яким запуск сервісів відбувається запуском однієї команди як на віддаленому сервері, так і локально.

Все, що потрібно мати користувачу — встановлений Docker, який підтримується Windows, Linux та macOS.

Далі необхідно перейти в папку з проектом і виконати дві команди:

1) docker-compose build;

2) docker-compose up.

Система автоматично створить віртуальне середовище чи ізольований віртуалізований контейнер, у якому встановить всі необхідні залежності й запустить у правильному порядку всі необхідні сервіси.

Усе, що необхідно потім зробити користувачеві, це перейти за посиланням http://localhost:3000/, якщо це локальний комп’ютер, або за адресою віддаленого сервера.

5.2 Налаштування системи

У правому верхньому кутку розміщується вкладка «Конфігурації», яка має два пункти: «Рівні вакансії» і «Технології» (рисунок 5.1).

Перший пункт меню рекомендується лише для ознайомлення, він вже налаштований. Технології можна не тільки переглянути, а й налаштувати. Наразі в системі прописано 171 технологія, багато з яких можуть бути зайвими, оскільки є досить специфічними й не можуть бути опрацьовані в рамках навчального процесу університету чи кафедри. Також можливо додавати нові технології, які виникатимуть з часом.

Сторінка конфігурації містить три секції. Перша секція надає можливість встановити значення за замовчуванням. Вибір надається серед технологій, вже наявних у системі. Поле вводу є автодоповненням, тому можна почати ввід і обрати з відфільтрованих варіантів. Якщо потрібно, тут можна вказати максимальні відстані у словах від обов'язкового правила. Для того, щоб задати саме правило, необхідно ввімкнути перемикач. Для того, щоб зберегти зміни, зроблені в цій секції, необхідно натиснути на кнопку «Зберегти».

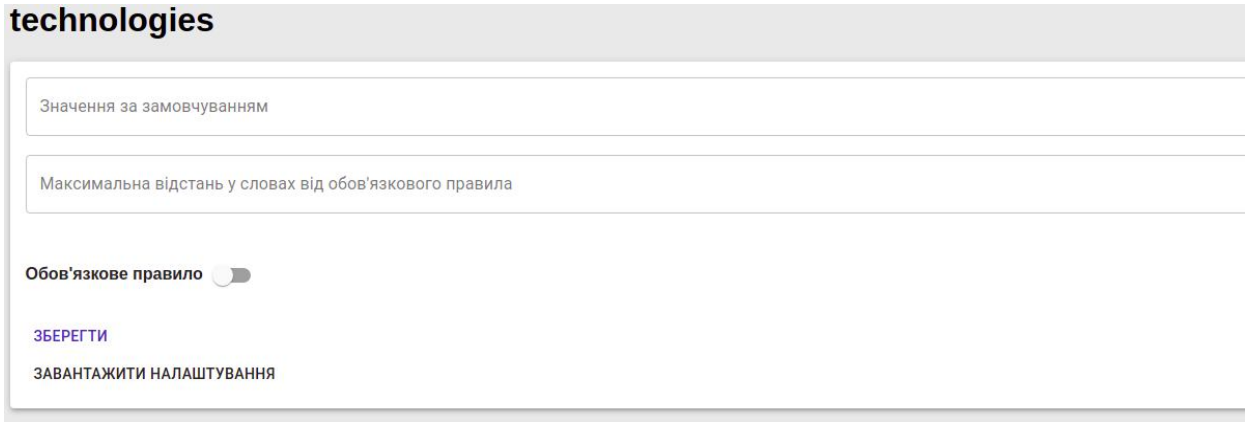


Рисунок 5.1 — Перша секція налаштувань на сторінці конфігурації технологій

Друга секція дає змогу випробувати описану конфігурацію й перевірити правильність налаштувань (рисунок 5.2). Все, що потрібно, це ввести текст і натиснути на кнопку «Тестувати», після чого з'являються мітки з усіма знайденими технологіями.

Наступна секція включає в себе поле вводу назви нової технології для того, щоб додати її в базу даних (рисунок 5.3). Після цього потрібно натиснути клавішу Enter. Трохи нижче з'явиться нова технологія, яку ще потрібно додати до системи.

Варто зазначити, що введена в це поле назва буде використовуватися скрізь у системі і має бути максимально зрозумілою користувачеві. Для прикладу, часто деякі технології відомі більше за абревіатурою назви, але записати тільки її буде помилковим. Для цих випадків для існуючих технологій застосовувався такий підхід: повна назва технології + пробіл + абревіатура в круглих дужках.

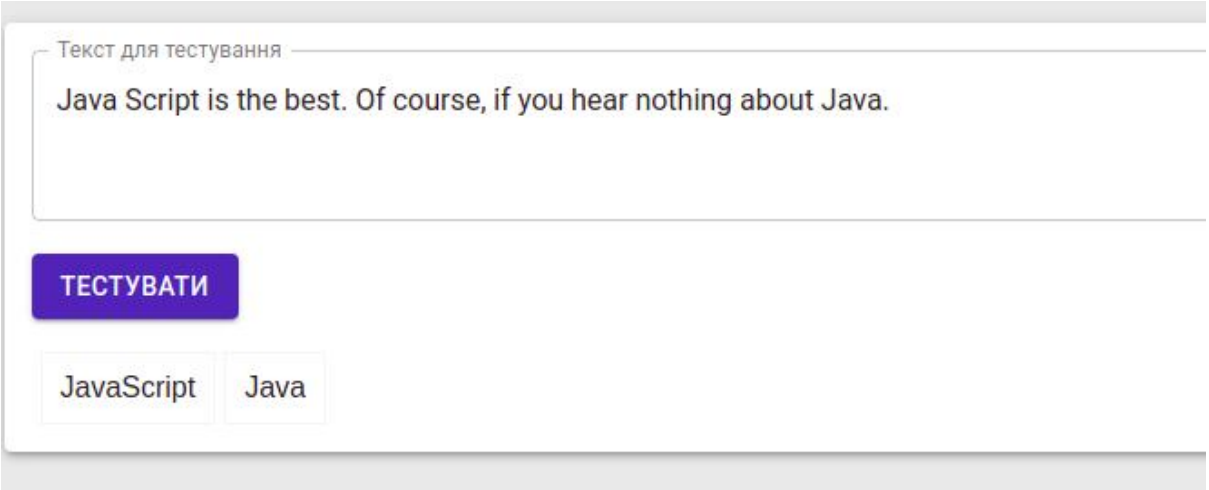


Рисунок 5.2 — Друга секція тестування конфігурації

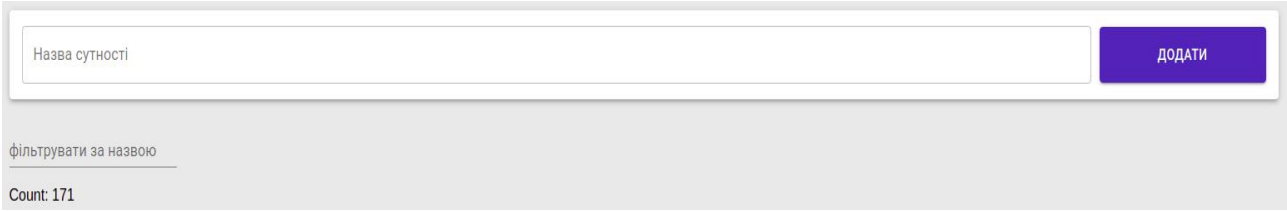


Рисунок 5.3 — Третя секція створення нових технологій

Також можна побачити фільтр, який дає можливість фільтрувати технології за назвою. Підхід, описаний вище, дає можливість знаходити технології як за повною назвою, так і за скороченням.

Далі йдуть вже наявні у системі технології (рисунок 5.4, 5.5). Тут можна змінювати назву технології, тип правила (рядок, регулярний вираз, усі правила, будь-яке правило) і конфігурацію правила. Перемикач регулює, чи використовувати цю технологію екстрактору, а іконка сміттєвої корзини видаляє технологію. Якщо користувач змінить хоч щось у технології, справа біля технології з’являється іконка

дискети, що дає можливість зберегти зміни. Тобто, зміну в кожну технологію необхідно зберігати окремо.

Одночасно на сторінці може бути відображено лише 20 технологій. Внизу сторінки можна побачити пагінацію (сторінки), що дають можливість побачити наступні 20 технологій.

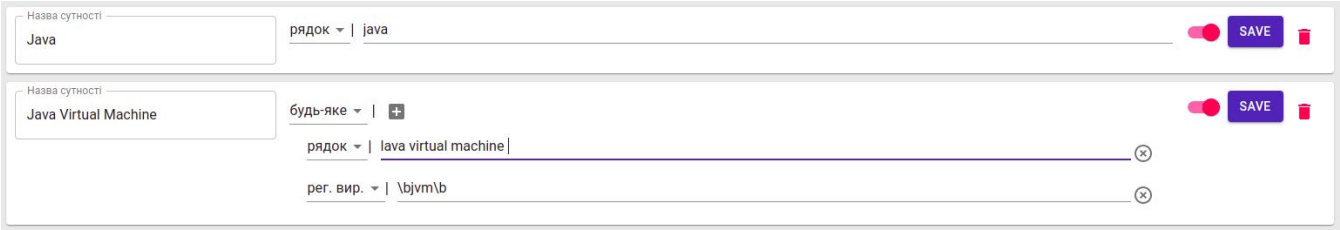


Рисунок 5.4 — Четверта секція переліку наявних технологій (приклад 1)

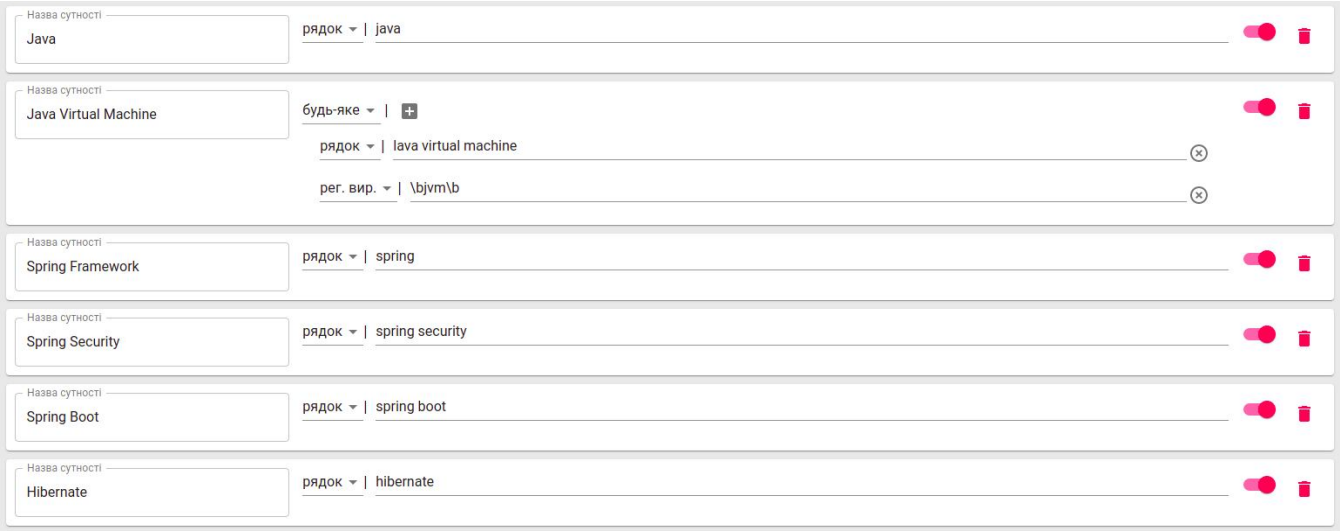


Рисунок 5.5 — Четверта секція переліку наявних технологій (приклад 2)

На рисунку 5.6 подано приклад того, який вигляд буде мати технологія після фільтрування. Як можна бачити, зображено тільки ті записи, які містять у собі слово «Java» без врахування регістру і незалежно від того, в якій позиції розміщується збіг.

Для зручності також додається інформаційний текст з кількістю записів після фільтрації.

При запуску системи, а також кожні наступні два дні відбувається автоматичний парсинг сайтів і в системі з’являються нові вакансії, які необхідно

підтвердити. Для цього необхідно перейти у вкладку «Сторінка адміністратора» та натиснути на єдину на цій сторінці кнопку «Затвердити нові вакансії» (рисунок 5.7).

Під час цієї операції буде автоматично знайдено дублікати, застосовано додаткові фільтри, такі як: відсіювання вакансій, де технологій більше за 15 чи менше за одну. У результаті користувач має побачити схоже повідомлення (рисунок 5.8).

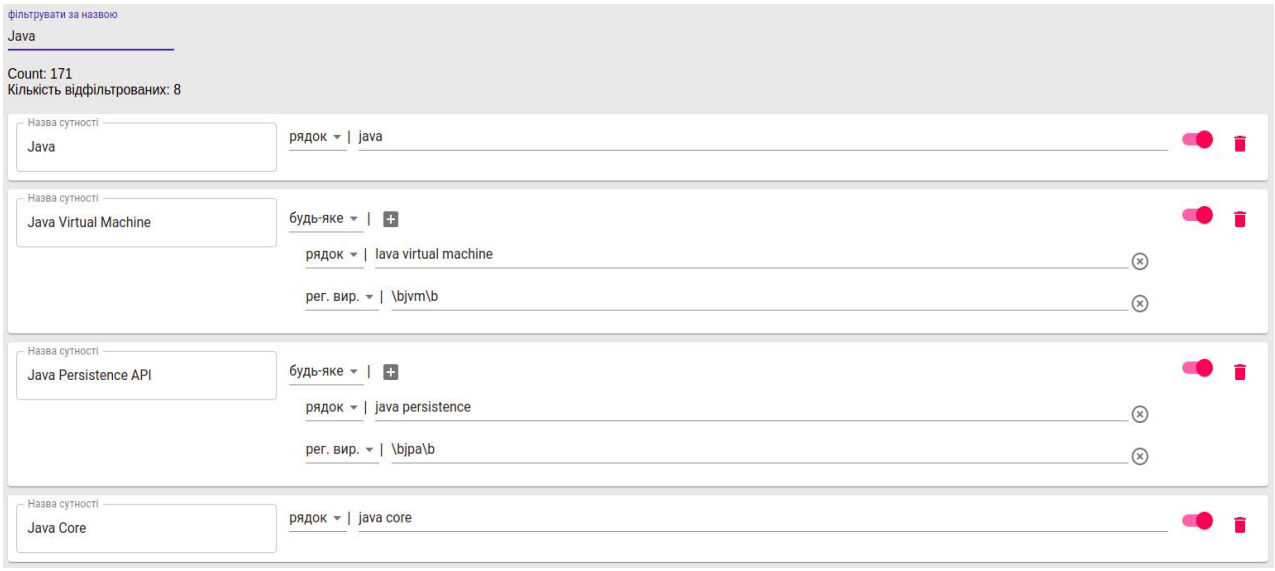


Рисунок 5.6 — Демонстрація функції фільтрування

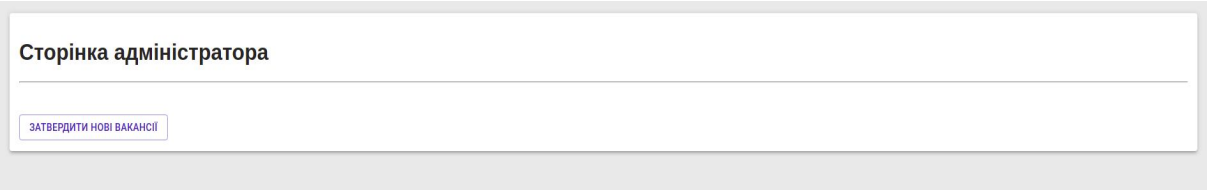


Рисунок 5.7 — Сторінка адміністратора

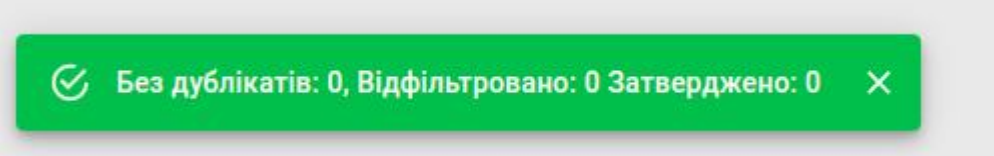


Рисунок 5.8 — Приклад повідомлення після затвердження нових вакансій

З наведених прикладів видно, що для роботи користувача з системою не потрібні спеціальні вміння.

5.3 Використання статистики

Сторінка статистики має дві колонки. Ліва містить фільтри, за якими буде будуватися статистика. Тут наявні два поля для введення з автодоповненням (рисунок 5.9).

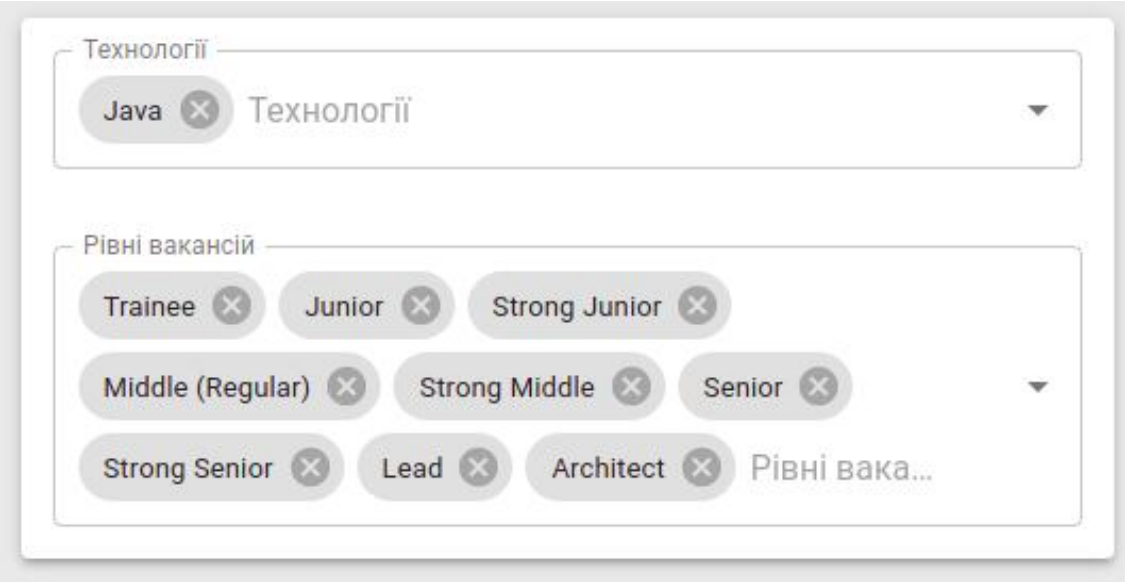


Рисунок 5.9 — Фільтри вакансій, за якими будується статистика

Друга колонка містить власне статистику. Якщо користувач не обрав жодної технології, то в правій колонці буде розміщуватися інформаційний елемент (рисунок 5.10).

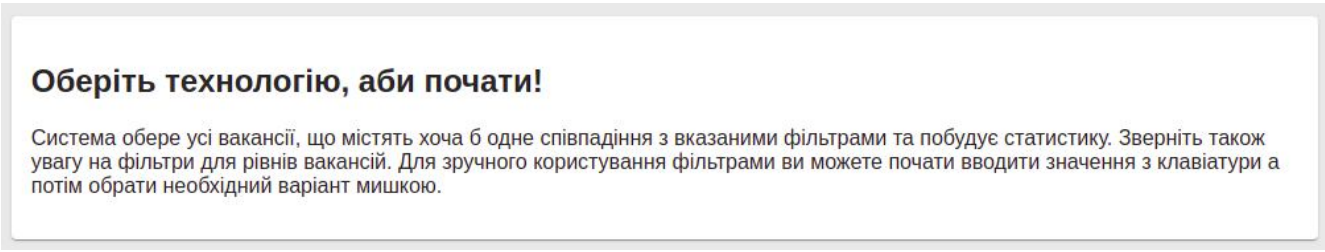


Рисунок 5.10 — Текст-підказка

Після того, як буде обрана хоча б одна технологія, користувач побачить на місці тексту-підказки розділи зі статистикою. Перш за все, з’явиться текстова інформація щодо загальної кількості вакансій, за якими буде відбуватися подальший

аналіз. Усі подальші приклади буде наведено для таких фільтрів: обрані усі можливі рівні вакансій, а також технологія Java.

Якщо для прикладу буде обрано інші налаштування, це буде явно вказано (рисунок 5.11).

Рисунок 5.11 — Текст, який інформує про кількість вакансій для аналізу

Першим інформаційним блоком є інформація про популярні стеки, які містять обрані у фільтрах технології. Таблиця містить три колонки: технології стеку (за винятком наведених у фільтрах), кількість вакансій з таким стеком, а також кількість вакансій з таким стеком з урахуванням підмножин технологій (рисунок 5.12).

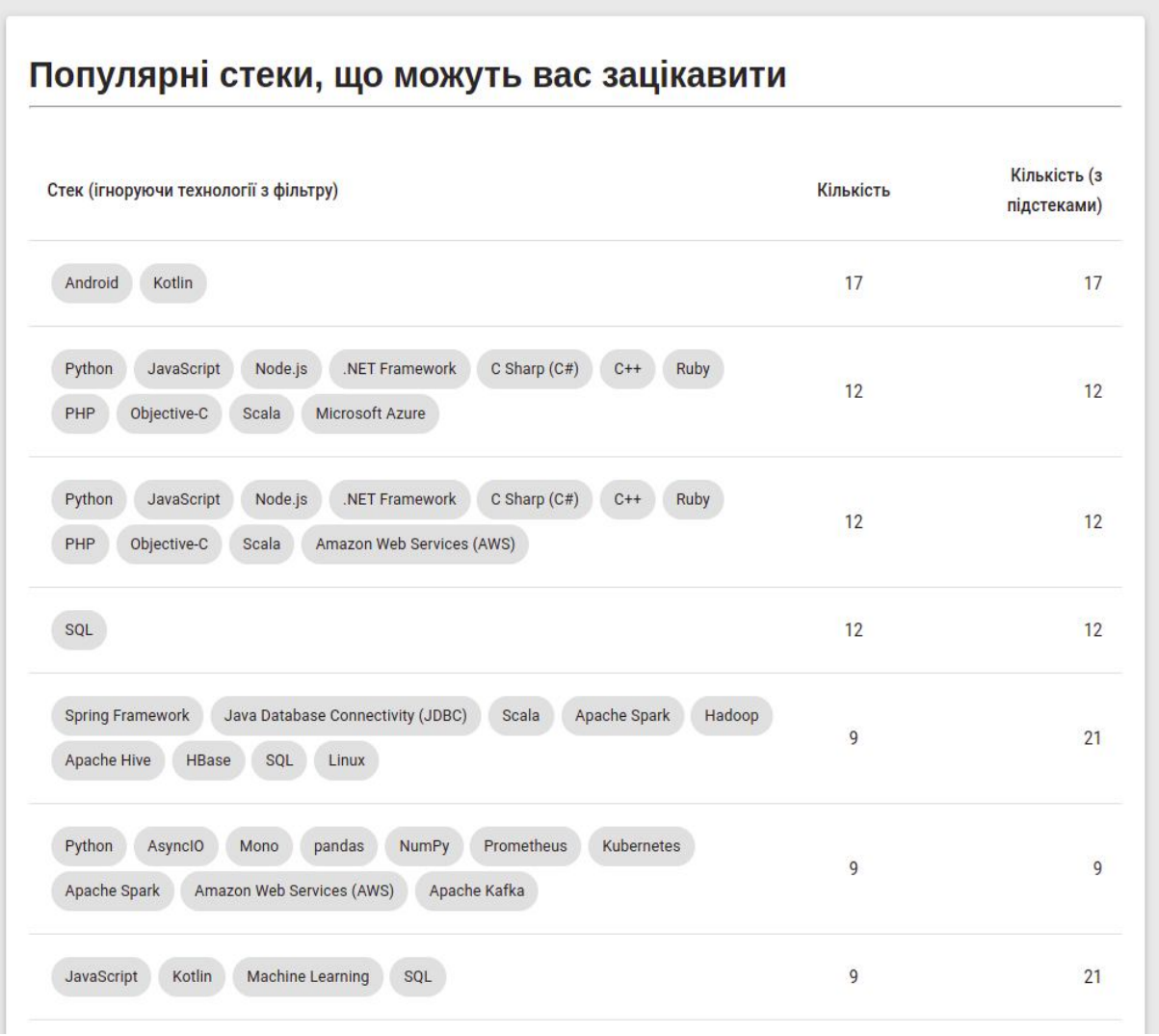
Така таблиця містить максимум 10 рядків. З неї можна отримати інформацію щодо тих технологій, які можуть бути максимально корисними для отримання конкретних посад. Важливою є саме можливість побачити закінчений технологічний стек, що дає інформацію щодо того, які сукупності технологій необхідні для виконання реальних завдань бізнесу. Як видно з зображення, прикладами популярних стеків є (Java, Android, Kotlin), що, швидше за все є мобільною розробкою на платформі Android, а також (Java, SQL), що може стосуватися тестувальників.

Наступна секція відображає популярність кожної технології, наявної в обраних вакансіях (рисунок 5.13). Це надає можливість також оцінити, які нові технології принесуть максимальну користь інженеру. З поданої інформації видно, що знання SQL і Python є досить важливими.

Наступна секція відображає частоту виникнення вакансій, пов’язаних з технологією у фільтрах (рисунок 5.14). Для кожної вакансії буде створено таку секцію. Всі дані агреговано по тижнях. Наразі в системі наявно даних приблизно за місяць, але в майбутньому можливо буде краще бачити тенденції популярності

технології з часом і робити висновки. Варто також згадати про можливість фільтрувати рівні вакансії, що дасть можливість розуміти тенденції, наприклад, для початківців. Це дає змогу, наприклад, кафедрі планувати, наскільки реально працевлаштувати студентів.

Остання секція відображає відношення кількості вакансій для кожного рівня (рисунок 5.15).

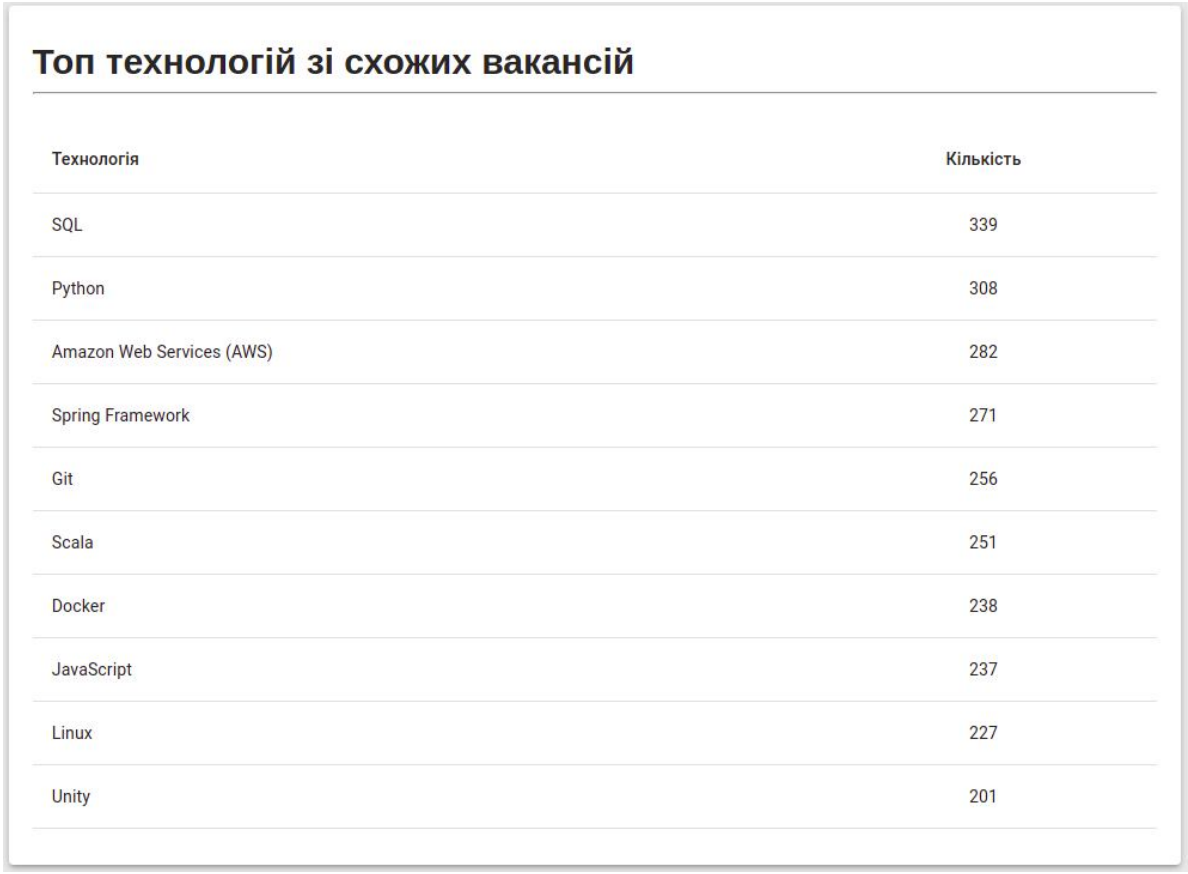


Популярні стеки, що можуть вас зацікавити

Стек (ігноруючи технології з фільтру)	Кількість	Кількість (з підстеками)
Android, Kotlin	17	17
Python, JavaScript, Node.js, .NET Framework, C Sharp (C#), C++, Ruby, PHP, Objective-C, Scala, Microsoft Azure	12	12
Python, JavaScript, Node.js, .NET Framework, C Sharp (C#), C++, Ruby, PHP, Objective-C, Scala, Amazon Web Services (AWS)	12	12
SQL	12	12
Spring Framework, Java Database Connectivity (JDBC), Scala, Apache Spark, Hadoop, Apache Hive, HBase, SQL, Linux	9	21
Python, AsyncIO, Mono, pandas, NumPy, Prometheus, Kubernetes, Apache Spark, Amazon Web Services (AWS), Apache Kafka	9	9
JavaScript, Kotlin, Machine Learning, SQL	9	21

Рисунок 5.12 — Секція популярних технологічних стеків

За допомогою цієї статистики можливо оцінити мінімально необхідний рівень англійської мови для технічного спеціаліста. Важливо зазначити, що, залежно від технології він може дещо змінюватися.



Топ технологій зі схожих вакансій

Технологія	Кількість
SQL	339
Python	308
Amazon Web Services (AWS)	282
Spring Framework	271
Git	256
Scala	251
Docker	238
JavaScript	237
Linux	227
Unity	201

Рисунок 5.13 — Інформаційна секція: Топ технологій у схожих вакансіях

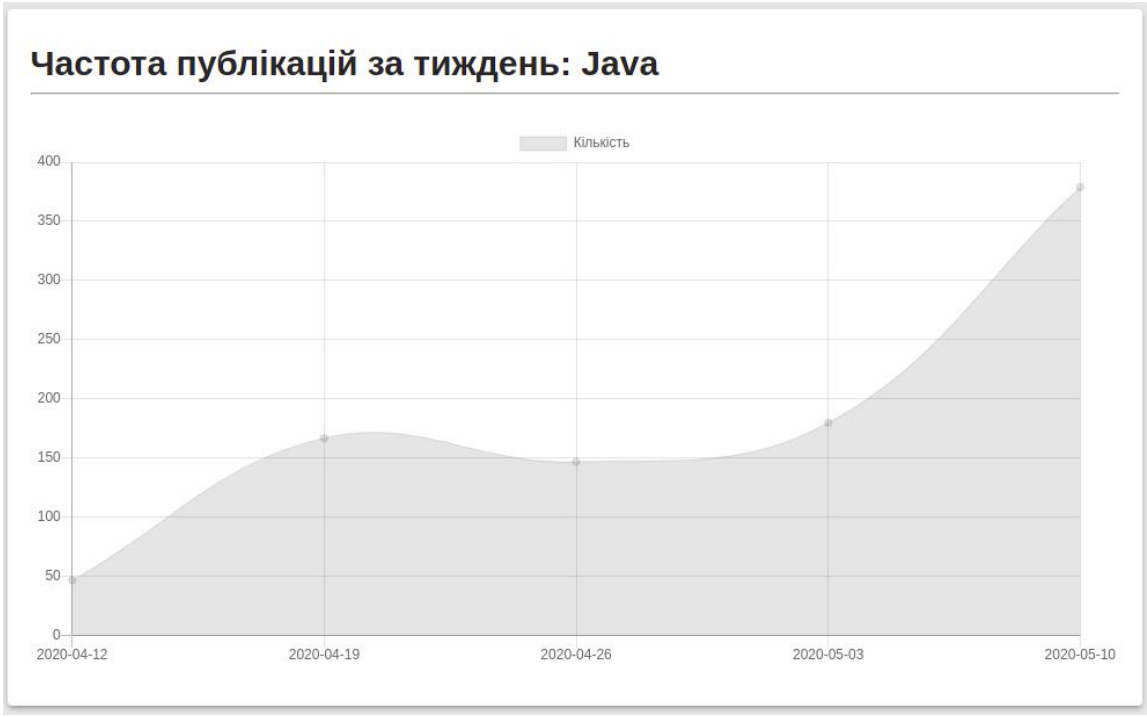


Рисунок 5.14 — Секція, що відображає частоту публікацій вакансій з певною технологією

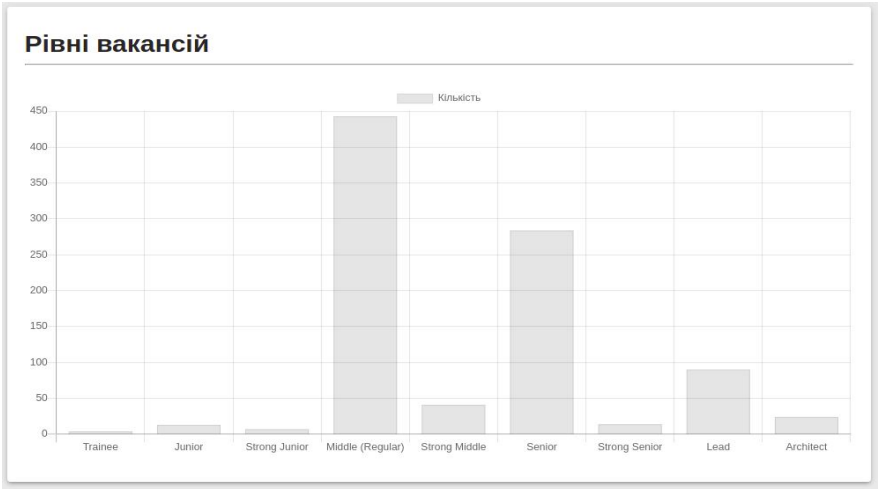


Рисунок 5.15 — Статистика розподілу рівня вакансій

Можна також помітити залежність розподілу рівня вакансій і рівня знання англійської мови. Здебільшого, більш високий рівень вакансії вимагає кращого володіння англійською. Також рівень може змінюватися з часом.

ВИСНОВКИ

При виконанні дипломної роботи було:

1) виконано аналіз існуючого ринку вакансій для інформаційних технологій, зокрема:

а) виявлено правила і закономірності;

б) визначено типову структуру вакансії;

в) визначено важливість і ступені довіри до інформації з різних джерел;

г) вибірано оптимальне джерело інформації;

2) проаналізовано існуючі рішення:

а) виявлено проблеми, які розв’язуються;

б) проаналізовано переваги й недоліки кожного підходу;

3) розроблено архітектуру сервісів та їхню взаємодію, встановлено формальні вимоги до кожного з них;

4) реалізовано й протестовано програмний код кожного сервісу:

а) реалізовано систему парсерів, які збирають дані з Інтернет-ресурсів;

б) реалізовано програмний модуль, який обробляє дані;

в) реалізовано сервер, який керує взаємодією інших модулів і керує зв’язком з базами даних;

г) реалізовано графічний інтерфейс для користувача та адміністратора;

5) створено конфігурацію для автоматичного розгортання застосунку й забезпечення комунікації між сервісами.

Кінцевий продукт дає змогу самостійно чи з допомогою експерта аналізувати певні вимоги й закономірності, приймати рішення щодо вивчення чи вдосконалення певних навичок інженера. Також може допомогти у формуванні навчального плану закладів освіти за умови консультації з експертами.

Результати роботи доповідалися на конференціях «Сучасні проблеми наукового забезпечення енергетики» [30] і «Сучасні аспекти розробки програмного забезпечення».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Robota.ua [Електронний ресурс] — Режим доступу до ресурсу: https://rabota.ua/

2. Work.ua [Електронний ресурс] — Режим доступу до ресурсу: https://www.work.ua/

3. Djinni.co [Електронний ресурс] — Режим доступу до ресурсу: https://djinni.co/

4. Dou.ua [Електронний ресурс] — Режим доступу до ресурсу: https://djinni.co/

5. Scrapy Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://docs.scrapy.org/en/latest/intro/tutorial.html.

6. Mitchell R. Web Scraping with Python / Ryan Mitchell. — O'Reilly Media, Inc, 2018. — 280 с.

7. Flask.User's Guide [Електронний ресурс] — Режим доступу до ресурсу: https://flask.palletsprojects.com/en/1.1.x/.

8. Fielding R. Architectural Styles and the Design of Network-based Software Architectures : дисертація докт. комп. наук [Електронний ресурс] / R. Fielding. — 2000. — 162 с. — Режим доступу до ресурсу: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

9. Гринберг М. Разработка веб-приложений с использованием Flask на языке Python / М. Гринберг. — М.: ДМК Пресс, 2014. — 272 с.

10. Бэнкер К. MongoDB в действии / К. Бэнкер. — М.: ДМК Пресс, 2014. — 394 с.

11. Дейт К. Введение в системы баз данных / К. Дейт. — М.: Вильямс, 2005. — 1328 с.

12. Ньюмен С. Создание микросервисов / С. Ньюмен. — СПб: Питер, 2016. — 304 с.

13. TIOBE Index [Електронний ресурс] — Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/

14. Clements P. Documenting Software Architectures: Views and Beyond. Second Edition. / P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. — Addison-Wesley Professional, 2010. — 540 p.

15. Бизли Д. Язык программирования Python. Справочник / Д. Бизли. — ДиаСофт, 2000. — 336 с.

16. Wesley J. Core Python Programming / J. Wesley. — Prentice Hall PTR, 2000. — 1136 p.

17. Лутц М. Программирование на Python / М. Лутц. — СПб.: Символ-Плюс, 2011. — 992 с.

18. Маккинли У. Python и анализ данных / У. Маккинли. — ДМК Пресс, 2015. — 482 с.

19. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series) / E.Gamma, R. Helm, R. Johnson, J. Vlissides. — Amazon Warehouse, 1994. — 395 с.

20. Редмонд Э. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL / Э. Редмонд, Д. Уилсон. — М.: ДМК Пресс, 2013. — 384 с.

21. Plugge E. The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing / E. Plugge, P. Membrey, T. Hawkins. — Apress, 2010. — 327 p.

22. Hows D. The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB, Third Edition / D. Hows, P. Membrey, E. Plugge, T. Hawkins. — Apress, 2015. — 376 с.

23. Грофф Д. SQL: полное руководство, 3-е издание / Д. Грофф, П. Вайнберг, Э. Оппель. — М.: Вильямс, 2014. — 960 с.

24. Фиайли К. SQL: Руководство по изучению языка. / К. Фиайли. — М.: Peachpit Press, 2003. — 456 с.

25. Paul H. Unit Test Frameworks / H. Paul. — Jenson Books Inc, 2004. — 304 с.

26. Lundh F. Python Standard Library / F. Lundh. — O'Reilly & Associates, 2001. — 304 p.

27. Bass L. Software Architecture in Practice, 3rd Edition / L. Bass, P. Clements, R. Kazman. — Addison Wesley, 2012. — 624 p.

28. Hoberman S. Data Modeling for MongoDB / S. Hoberman. — Technics Publications, 2014. — 226 p.

29. Chodorow K. MongoDB: The Definitive Guide, 2nd Edition / K. Chodorow. — O'Reilly, 2013. — 432 p.

30. Бочок В.О. Система надання статистики про інженерні вакансії в IT / В.О. Бочок, Л.І. Кублій // Сучасні проблеми наукового забезпечення енергетики: Матеріали XVIII Міжнародної науково-практичної конференції молодих вчених і студентів 2020 року. — К.: КПІ ім. Ігоря Сікорського, 2020. — Т. 2. — С. 120.

ДОДАТОК А

Сервіс live-статистики стану IT-ринку на певний момент чи проміжок часу

Специфікація

УКР.НТУУ“КПІ ім. Ігоря Сікорського”.ТВ6124_20Б

Аркушів 2

2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ «КПІ ім. Ігоря Сікорського».ТВ6124_20Б 81-1	Записка	Пояснювальна записка
Компоненти		
УКР.НТУУ «КПІ ім. Ігоря Сікорського».ТВ6124_20Б 12-1	Текст програмного модуля	Текст модуля обробки даних
УКР.НТУУ «КПІ ім. Ігоря Сікорського».ТВ6124_20Б 13-1	Опис програми	

ДОДАТОК Б

Сервіс live-статистики стану IT-ринку на певний момент чи проміжок часу

Текст програмного модулю

УКР.НТУУ“КПІ ім. Ігоря Сікорського”.ТВ6124_20Б 12-1

Аркушів 10

2020

```
from abc import ABCMeta, abstractmethod
from src.data_processors.utils import is_entry_subset, calculate_distance_words
import re


class RuleResponse:
    def __init__(self, result: bool, entries: list = None):
        self.__result = result
        # entries -> list of tuples, like (start_position, end_position)
        self.__entries = entries

    @property
    def result(self) -> bool:
        return self.__result

    @property
    def entries(self) -> list:
        return self.__entries


class Rule:
    __metaclass__ = ABCMeta

    @abstractmethod
    def check(self, text: str) -> RuleResponse:
        """Checks if the text matches the rule and returns RuleResponse"""


class StringRule(Rule):
    _pattern = None
    _pattern_len = None

    def __init__(self, pattern: str):
        self._pattern = pattern
        self._pattern_len = len(pattern)

    def check(self, text: str) -> RuleResponse:
        start = 0
        entries = []
        while True:
            start = text.find(self._pattern, start)
```

```
            if start == -1:
                break
            entries.append((start, start + self._pattern_len))
            start += 1

        if len(entries) > 0:
            return RuleResponse(True, entries)
        else:
            return RuleResponse(False)




class RegexRule(Rule):
    _pattern = None

    def __init__(self, pattern: str):
        self._pattern = re.compile(pattern)

    def check(self, text: str) -> RuleResponse:
        entries = []
        for match in self._pattern.finditer(text):
            entries.append(
                (match.start(), match.end()))
        if len(entries) > 0:
            return RuleResponse(True, entries)
        else:
            return RuleResponse(False)


class AllRule(Rule):
    _rules = []
    _max_distance_words = None

    def __init__(self, rules, max_distance_words=None):
        self._rules = rules
        self._max_distance_words = max_distance_words

    def add_rule(self, rule):
        self._rules.append(rule)

    def check(self, text) -> RuleResponse:
        entries = []
```

```
        for rule in self._rules:
            rule_response = rule.check(text)
            if not rule_response.result:
                return RuleResponse(False)
            entries.append(rule_response.entries)

        if len(entries) < 0:
            return RuleResponse(False)

        # get all combinations
        combined_entries = entries.pop()
        for i in range(len(entries)):
            combined_entries = self._combine_entries(text, combined_entries,
entries[i])

        combined_entries.sort(key=lambda x: x[0])  # sort by start index
        combined_entries_mask = [True for x in combined_entries]

        for i in range(len(combined_entries) - 1):
            if not combined_entries_mask[i]:
                continue
            for j in range(i + 1, len(combined_entries)):
                if not combined_entries_mask[j]:
                    continue
                if is_entry_subset(combined_entries[j], combined_entries[i]):
                    combined_entries_mask[i] = False
                    break
                elif is_entry_subset(combined_entries[i], combined_entries[j]):
                    combined_entries_mask[j] = False

        # filter entries that are superset for other
        combined_entries = [combined_entries[i] for i in range(len(combined_entries))
if combined_entries_mask[i]]

        if len(combined_entries) < 1:
            return RuleResponse(False)

        return RuleResponse(True, combined_entries)

    def _combine_entries(self, text: str, first_list: list, second_list: list) ->
list:
        entries = []
        for f_entry in first_list:
```

```
            for s_entry in second_list:
                # skip entries if words distance bigger than defined
                if is_entry_subset(f_entry, s_entry) or is_entry_subset(
                  s_entry, f_entry):
                    continue
                if (self._max_distance_words is not None and
                        calculate_distance_words(text, f_entry, s_entry) >
                    self._max_distance_words):
                    continue
                entries.append(
                    (min([f_entry[0], s_entry[0]]), max([f_entry[1], s_entry[1]]))
                )
        return entries


class AnyRule(Rule):
    _rules = []

    def __init__(self, rules: list):
        self._rules = rules

    def check(self, text: str) -> RuleResponse:
        entries = []
        for rule in self._rules:
            rule_response = rule.check(text)
            if rule_response.result:
                entries.extend(rule_response.entries)
        if len(entries) < 1:
            return RuleResponse(False)
        return RuleResponse(True, entries)




from src.data_processors.rules import Rule, AllRule, AnyRule, StringRule, RegexRule
from src.data_processors.utils import is_entry_subset, is_entries_have_interception,
calculate_distance_words
from itertools import combinations


class RuleBuilder:
```

```
    @staticmethod
    def build_rule(config_rule: dict):
        if config_rule['type'] == 'all':
            return RuleBuilder._build_all_rule(config_rule)
        elif config_rule['type'] == 'any':
            return RuleBuilder._build_any_rule(config_rule)
        elif config_rule['type'] == 'string':
            return RuleBuilder._build_string_rule(config_rule)
        elif config_rule['type'] == 'regex':
            return RuleBuilder._build_regex_rule(config_rule)

    @staticmethod
    def _build_all_rule(config):
        return AllRule(
            [RuleBuilder.build_rule(rule_config) for rule_config in config['value']],
            config.get('max_distance_words')
        )

    @staticmethod
    def _build_any_rule(config):
        return AnyRule([RuleBuilder.build_rule(rule_config) for rule_config in
config['value']])

    @staticmethod
    def _build_string_rule(config):
        return StringRule(config['value'])

    @staticmethod
    def _build_regex_rule(config):
        return RegexRule(config['value'])


class ExtractorMessage:
    # None - undefined,
    # 1 - info
    # 2 - warning
    # 3 - error
    _level: int
    _message: str
    _details: str

    INFO_LEVEL = 1
    WARNING_LEVEL = 2
```

```
    ERROR_LEVEL = 3

    @staticmethod
    def info(message: str, details: str = None):
        return ExtractorMessage(ExtractorMessage.INFO_LEVEL, message, details)

    @staticmethod
    def warning(message: str, details: str = None):
        return ExtractorMessage(ExtractorMessage.WARNING_LEVEL, message, details)

    @staticmethod
    def error(message: str, details: str = None):
        return ExtractorMessage(ExtractorMessage.ERROR_LEVEL, message, details)

    def __init__(self, level: int, message: str, details: str = None):
        self._level = level
        self._message = message
        self._details = details

    @property
    def level(self) -> int:
        return self._level

    @property
    def message(self) -> str:
        return self._message

    @property
    def details(self) -> str:
        return self._details


class ExtractorResponse:
    _messages: tuple
    _entries: tuple

    def __init__(self, entries: tuple, messages: tuple = None):
        self._entries = entries
        self._messages = messages

    @property
    def messages(self) -> tuple:
        return self._messages
```

```
    @property
    def entries(self) -> tuple:
        return self._entries


class Extractor:
    _default = None
    # {key: rule: Rule}
    _rule_holders: dict
    # Used to define which word constructions should be present in the text
    # @see abstract Rule
    _required_rule: Rule = None
    # maximum inclusive distance from required word constructions entries
    _required_rule_max_word_distance: int = None

    def __init__(self, config):
        self._default = config.get('default')
        if config.get('required_rule'):
            self._required_rule = RuleBuilder.build_rule(config['required_rule'])
        if 'required_rule_max_word_distance' in config:
            self._required_rule_max_word_distance =
config['required_rule_max_word_distance']
        self._rule_holders = {
            rule_holder['key']: RuleBuilder.build_rule(rule_holder['rule'])
            for rule_holder in config['rule_holders']
        }

    def extract(self, text: str) -> ExtractorResponse:
        lower_text = text.lower()

        log = []

        required_rule_entry_list = []
        if self._required_rule is not None:
            required_rule_response = self._required_rule.check(lower_text)
            if not required_rule_response.result:
                return ExtractorResponse(None)
            required_rule_entry_list = required_rule_response.entries

        key_entry_list = self._get_key_entry_list(lower_text)
        key_entry_list = self._filter_subsets(key_entry_list)
```

```
        key_entry_list =
self._filter_entries_by_required_rule_max_distance(lower_text,

key_entry_list,

required_rule_entry_list)
        for f, s in combinations(key_entry_list, 2):
            if is_entries_have_interception(f[1], s[1]):
                log.append(ExtractorMessage.warning(
                    "Entries interception", f"Between '{f[0]}' and '{s[0]}"))

        filtered_keys = self._extract_keys_and_filter_duplicates(key_entry_list)

        if len(filtered_keys) != len(key_entry_list):
            log.append(ExtractorMessage.info("Keys duplications"))

        if len(filtered_keys) < 1 and self._default is not None:
            return ExtractorResponse(tuple([self._default]), tuple(log))
        return ExtractorResponse(tuple(filtered_keys), tuple(log))

    def _filter_entries_by_required_rule_max_distance(self, text: str,
                                                      key_entry_list: list,
required_rule_entry_list: list) -> list:
        if self._required_rule_max_word_distance is not None:
            key_entry_list_mask = [False for _ in key_entry_list]
            for required_rule_entry in required_rule_entry_list:
                for i in range(len(key_entry_list)):
                    if key_entry_list_mask[i]:
                        continue
                    if calculate_distance_words(
                            text, required_rule_entry, key_entry_list[i][1]
                    ) <= self._required_rule_max_word_distance:
                        key_entry_list_mask[i] = True
            return [key_entry_list[i] for i in range(len(key_entry_list)) if
key_entry_list_mask[i]]
        return key_entry_list

    def _get_key_entry_list(self, text: str) -> list:
        key_entry_list = []
        for key, rule in self._rule_holders.items():
            rule_response = rule.check(text)
            if not rule_response.result:
                continue
```

```
            for entry in rule_response.entries:
                key_entry_list.append(
                    (key, entry)
                )
        return key_entry_list

    @staticmethod
    def _filter_subsets(key_entry_list: list) -> list:
        key_entry_list.sort(key=lambda x: x[1][0])  # by start_pos
        key_entry_list_mask = [True for x in key_entry_list]

        for i in range(len(key_entry_list) - 1):
            if not key_entry_list_mask[i]:
                continue
            _, f_entry = key_entry_list[i]
            for j in range(i + 1, len(key_entry_list)):
                if not key_entry_list_mask[j]:
                    continue
                _, s_entry = key_entry_list[j]
                if is_entry_subset(f_entry, s_entry):
                    key_entry_list_mask[i] = False
                    break
                elif is_entry_subset(s_entry, f_entry):
                    key_entry_list_mask[j] = False

        return [key_entry_list[i] for i in range(len(key_entry_list)) if
key_entry_list_mask[i]]

    @staticmethod
    def _extract_keys_and_filter_duplicates(key_entry_list: list) -> list:
        return list(set([key for key, _ in key_entry_list]))
```

ДОДАТОК В

Сервіс live-статистики стану IT-ринку на певний момент чи проміжок часу

Опис програмного модулю

УКР.НТУУ“КПІ ім. Ігоря Сікорського”.ТВ6124_20Б 13-1

Аркушів 5

2020

АНОТАЦІЯ

Метою роботи було створення модуля, що буде здатний згідно із заданою конфігурацією виділяти усі наявні ключові слова з тексту довільного формату та структури. Програма здатна вирішувати колізії під час обробки та гнучко конфігуруватися. Наразі використовується для витягання з тексту технологій а також рівня вакансії. В перспективі код може бути використаний також для інших цілей при умові наявності відповідної конфігурації.

ЗМІСТ

В.1 ВІДОМОСТІ ПРО ПРОГРАМНИЙ МОДУЛЬ

В.1.1 Загальні відомості

Даний програмний модуль було розроблено мовою Python3.6. Наразі код являє собою набір класів, що об'єднані у модуль. Під час розробки використовувалися лише бібліотеки, що наявні у стандартній збірці Python.

В.1.2 Функціональне призначення

Програма призначена для знаходження ключових слів у тексті довільної форми за конфігурацією, що задається у вигляді JSON-об’єкту, що перетворюється програмою на конфігурацію з внутрішніх функціональних класів.

В.1.3 Опис логічної структури

Було розроблено модуль, головним класом якого є клас Extractor. Він приймає конфігурації словником типу {ключове слово => клас Rule}. Вся конфігурація відбувається за допомогою похідних від Rule класів, а саме: StringRule, RegexRule, AllRule, та AnyRule. Кожен з них має свою конфігурацію. StringRule та RegexRule є простими правилами, а AllRule та AnyRule - комплексними. Комплексні правила можуть містити в середині себе довільну кількість та глубину вкладень. Саме ці класи відповідають за пошук співпадінь у тексті. До того ж на AllRule покладена додаткова логіка фільтрації співпадінь для вирішення колізій. На клас Extractor також покладена роль фільтрації співпадінь та вирішення колізій.

В.1.4 Використані технічні засоби

Код може бути запущений на будь-якому сучасному комп’ютері. Під час розробки використовувався комп’ютер з Intel® Core™ i5-8265U CPU @ 1.60GHz × 8, 8Gb RAM, Mesa Intel® UHD Graphics 620 (WHL GT2).

В.1.5 Виклик і завантаження

Код реалізує програмний модуль Python, а отже може бути імпортований та використаний як бібліотека у будь-якому файлі Python 3.6. Після ініціалізації вимагає наявності інтерпретатору версії 3.6 чи більше.

В.1.6 Вхідні та вихідні дані

Вхідними даними до ініціалізованого модулю є текст, вихідними ж масив ключових слів, що знайдені у ньому. Ключові слова не повторюються, пошук регістронезалежний.

ДОДАТОК Г

Сервіс live-статистики стану IT-ринку на певний момент чи проміжок часу

АПРОБАЦІЯ

Міжнародна науково-практична конференція «Сучасні проблеми наукового забезпечення енергетики: Матеріали XVIII Міжнародної науково-практичної конференції молодих вчених і студентів 2020 року», м. Київ, 2020

УКР.НТУУ“КПІ ім. Ігоря Сікорського”.ТВ6124_20Б

Аркушів 4

2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ

Матеріали XVIII Міжнародної науково-практичної конференції молодих вчених і студентів 2020 року

ТОМ 2

Київ- 2020

УДК 004.622:519.255

Студент 4 курсу, гр. ТВ-61 Бочок В.О.
Доц., к.т.н. Кублій Л.І.

СИСТЕМА НАДАННЯ СТАТИСТИКИ ПРО ІНЖЕНЕРНІ ВАКАНСІЇ В ІТ

Сучасні інформаційні технології (ІТ) постійно розвиваються і змінюються. Також змінюються потреби бізнесу й інструменти, які задовольняють ці потреби. Можна нарахувати багато напрямів, які часто взаємодіють один з одним: веб-розробка (back-end і front-end), мобільна розробка (android, iOS, Windows), розробка десктопних застосунків, big data, data science, data engineering, analytics і багато інших. Кожен з напрямів має розділи і підрозділи і кожен з них потребує своїх специфічних знань і технологій.

Наявність великої кількості напрямів і необхідних сукупностей технологій приводить до неможливості оцінити, наскільки спеціаліст є актуальним і які додаткові технології найбільш підвищать цінність інженера для роботодавця. Звичайно ж, існує велика кількість статей і блогів, які подають своє персональне бачення цієї проблеми, але воно є, як правило, лише баченням автора і тому існує велика кількість суперечливих думок. Вже зайняті інженери можуть дізнаватися актуальну інформацію зі свого оточення, але це також не завжди універсальне бачення ситуації. Найкращий з доступних інструментів — сайти з вакансіями, де інженер може дізнатися, скільки вакансій для нього підходять і яких знань йому не вистачає. Цей метод найкращий, оскільки не містить суб'єктивної думки, а лише відображає реальні потреби бізнесу, проте він вимагає значної витрати часу і ручної фіксації прийнятних пропозицій.

Комп'ютерні системи здатні опрацьовувати не десятки вакансій, а тисячі за набагато менший проміжок часу, а також подавати результат в інформативнішому вигляді. Аналіз існуючих рішень показує, що деякі подібні системи вже є у сайтів вакансій work.ua, djinni.com, dou.ua тощо. Але вони надають не настільки детальну інформацію або ж використовують опитування людей, що не завжди дає об'єктивну інформацію.

Визначивши проблему і можливі джерела інформації, треба чітко продумати, яка інформація необхідна і яка доступна. Крім того, існує й інша проблема — вільний формат опису вимог до вакансії, що значно ускладнює виокремлення структурованих даних і підвищує ризик помилки при машинному опрацюванні. Також сайти багатомовні, можливі варіації навіть в написанні міста, для якого вакансія є актуальною (наприклад, Київ може бути передано як Kyiv або в російськомовних ресурсах як Киев).

Розв'язанням проблеми є створення комплексного конвеєра даних, який складається з виробника даних (Python Srapy [1]), виокремлювачів даних, які виокремлюють ключові слова за словником (технології, рівень англійської, необхідний досвід, рівень вакансії), трансформера даних, який перетворює слова до одного з заданих або повертає порожнє значення (використовуються для приведення назв міст до одного заданого словника), сервісу контролю якості (система логування, а також веб-інтерфейс, який дає змогу змінювати чи видаляти неякісні дані) і база даних (MongoDB), яка здатна зберігати різні формати даних і не вимагає утворювати постійну схему документів. База даних MongoDB також дає можливість виконувати складні запити на агрегування, фільтрування й групування даних для побудови висновків, які побачить користувач.

Таким чином, розроблена система бере дані з сайтів, виокремлює з них корисну інформацію, перетворює її до одного вигляду і записує до бази даних. За зібраною інформацією будуються інформаційні сторінки вакансій.

Перелік посилань:

1. Scrapy Tutorial. — https://docs.scrapy.org/en/latest/intro/tutorial.html